A MULTISENSORY OBSERVER MODEL FOR HUMAN SPATIAL ORIENTATION PERCEPTION

by

Michael C. Newman

B.S., Mechanical Engineering

Villanova University, 2007

Submitted to the Department of Aeronautics and Astronautics
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2009
[JUNE]

Signature of Author ..
Department of Aeronautics and Astronautics
May 24, 2009

Certified by .....
Charles M. Oman, Ph.D.
Senior Lecturer and Senior Research Engineer
Thesis Supervisor

Accepted by .....
Professor David L. Darmofal
Associate Department Head Chair, Committee on Graduate Students

A MULTISENSORY OBSERVER MODEL FOR HUMAN SPATIAL ORIENTATION PERCEPTION

by Michael C. Newman

Submitted to the Department of Aeronautics and Astronautics
on May 24, 2009 in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aeronautics and Astronautics

ABSTRACT: Quantitative "observer" models for spatial orientation and eye movements have been developed based on 1-G data from humans and animals (e.g. Oman 1982, 1991, Merfeld, et al 1993, 2002; Haslwanter 2000, Vingerhoets 2006). These models assume that the CNS estimates "down", head angular velocity and linear acceleration utilizing an internal model for gravity and sense organ dynamics, continuously updated by sensory-conflict signals. CNS function is thus analogous to a Luenberger state observer in engineering systems. Using a relatively small set of free parameters, Observer orientation models capture the main features of experimental data for a variety of different motion stimuli.

We developed a Matlab/Simulink based Observer model, including Excel spreadsheet input capability and a GUI to make the model accessible to less expert Matlab users. Orientation and motion predictions can be plotted in 2D or visualized in 3D using virtual avatars. Our Observer's internal model now computes azimuth, and pseudointegrates linear motion in an allocentric reference frame (perceived north-east-down). The model mimics the large perceptual errors for vertical motion observed experimentally. It retains the well validated "vestibular core" of the Merfeld perceptual model and predicts responses to angular velocity and linear accelerations steps, dumping, fixed radius centrifugation, roll tilt and OVAR. This model was further extended to include static and dynamic visual sensory information from four independent visual sensors (Visual Velocity, Position, Angular Velocity and Gravity/"Down").

Visual additions were validated against the Borah et al (1978) Kalman filter simulation results and validation data sets (Earth vertical constant velocity rotation in the light, somatogravic illusion in the light, and linear and circular vection). The model predicts that circular vection should have two dynamic components, and the recent finding of Tokumaru et al (1998) that visual cues influence somatogravic illusion in ways not accounted for by the Borah model. The model also correctly predicts both the direction of Coriolis illusion, and the magnitude of the resulting tilt illusion. It also predicts that the direction and mechanism of Pseudo-Coriolis illusion is fundamentally different from Coriolis, a prediction verified by means of a pilot experiment. Finally, the model accounts for the dynamics of astronaut post-flight tilt-gain and OTTR vertigos in ways not explained by previous static analyses (e.g. Merfeld, 2003).
Supported by the National Space Biomedical Research Institute through NASA NCC 9-58.

Thesis Supervisor: Charles M. Oman, Ph.D.
Title: Senior Lecturer and Senior Research Engineer

ACKNOWLEDGEMENTS

I would first and foremost like to thank Dr. Charles Oman for being my thesis advisor and providing me with the opportunity to work and research in the Man Vehicle Laboratory. The countless hours we spent brainstorming over chalkboard diagrams and hunched over computer simulations were monumental to the success of both this project and my time at MIT. Your work and dedication is sincerely appreciated.

A lot of people have contributed their time, ideas, feedback and support over the last two years of this project Thank you, Mark Brehon, Torin Clark, Dr. Thomas Jones, John Keller, Andy Liu, Dr. Dan Merfeld, Andrew Rader, Pierre Selva, Ron Small, Alexander Stimpson, Dr. Chris Wickens, and Dr. Larry Young.

To my friends and family who have supported and helped me along the way, thank you for keeping me sane and grounded in this endeavor. There are far too many of you to list, but I think you all know who you are.

I would finally like to thank the National Space Biomedical Research Institute who supported this project (Project SA1302 R.Small, PI) through NASA NCC 9-58.

CONTENTS

CHAPTER 1. INTRODUCTION

Mathematical models for human dynamic spatial orientation based on concepts from estimation theory in engineering have been widely applied in the field of vestibular physiology over the past several decades. Borah, et al (1978) proposed a Kalman Filter (KF) model describing how semicircular canal (SCC), otolith, and visual velocity sensory cues combined to yield an orientation perception. Borah's model utilized linear optimal estimation theory, but the inherent small angle assumptions precluded its use in many of the most important research applications. To circumvent this, Pommellet (1990) and Bilien (1993) developed Extended Kalman Filter (EKF) versions of the Borah et al model. However numerical instabilities limited application, and the work remained unpublished. McGrath (B. McGrath, personal communication) utilized KF models in the analysis of several US Navy aircraft accidents, but the complexity of KF model code and theory made them inaccessible to most accident investigation practitioners. A decade later, Merfeld et al (1993) proposed a model for SCC and otolith cue interaction in the generation of monkey eye movements. Merfeld's model used a nonlinear, Quaternion based approach to represent head orientation relative to gravity. It retained the "internal dynamic model" notion inherent in a KF, but employed a more ad-hoc scheme to correct internal dynamic model predictions using weighted sensory cues. However, Merfeld's model successfully matched a variety of experimental data using only four free parameters that served as "sensory conflict" weighting factors. Merfeld et al described their model as an "Observer", since that term is often used in engineering to describe an internal model state estimator whose expected vs. actual state measurement weighting coefficients are empirically derived; Luenberger, (1971). Subsequently Merfeld and Zupan (2002) extended the model to predict human eye movements, and Haslwanter et al (2000) and Vingerhoets, et al (2006, 2007) successfully applied it to human perception during off vertical axis rotation. Groen et al (2007) also proposed an alternative (non-Observer) model for human orientation perception, and provided a toolbox of Matlab routines for simulation. Small et al (2006) of Alion Corp. developed a Spatial Disorientation Analysis Tool (SDAT) that included sophisticated user interface, and rules for classification of classic orientation illusions, but that incorporated only a rudimentary model for sensory cue interaction. Unfortunately, these previous models all had limitations which precluded their application beyond the immediate research domain, e.g. in aircraft accident investigation or human spaceflight: 1) most modeling tools lacked a graphical user interface and visualization tools, so they could be operated by non-specialists. 2) the Observer models did not incorporate visual inputs, or predict perceived azimuth 3) the model predictions had not been compared with data from a broad set of experiments.4) the models did not account for the known inability of humans to correctly perceive large amplitude vertical motion, and 5) the models could not be readily adapted to mimic orientation in altered G environments. The goal of this thesis was to extend the Merfeld Observer model to remedy these deficiencies. The research was supported as part of a collaborative NSBRI funded research project with Alion Science and Technology, Inc, so a secondary goal was to produce a model which could ultimately be incorporated into their SDAT analysis tool.

CHAPTER 2. BACKGROUND

Borah, Young and Curry (1978 1988) originally developed a steady state KF to model the orientation perception of a human riding passively in a vehicle. Their model included vestibular motion cues as well as dynamic angular and linear visual velocity information. It was capable of predicting responses to a number of vestibular and visual-vestibular motion paradigms including circular vection, yaw rotation in the light, and sustained linear acceleration. Due to the linear nature of the implemented Kalman Filter, these predictions were however restricted to small head deviations from the postural upright. Pommellet and colleagues (1990) modified Borah's internal model and implemented a time varying Extended Kalman Filter to account for full non-linear motion dynamics. Although the Borah and Pommellet models both derived their "Kalman Gain" weighting coefficients based on optimal estimation theory, they assumed values for process (vehicle disturbance) and measurement (sensory noise) covariance ratios and bandwidths. However the values of these stochastic parameters were adjusted until the models fit data, and no physical basis for them was offered. Hence the ultimate basis for these KF models remained empirical. While Borah's model responses qualitatively matched perceptions for simple stimuli with the head near the erect position, the Pommellet EKF model results exhibited numerical instabilities in quaternion estimation, particularly for the more complex centrifuge profiles involving larger estimated tilts. A follow up EKF study by Bilien (1993) investigating the vestibular portions of the model encountered similar difficulties when modeling Coriolis responses.

Meanwhile, Merfeld et al (1993) utilized an "Observer" approach to propose a one dimensional model for the slow phase of the angular vestibule-ocular reflex (VOR) in response to yaw rotation about the gravitational vertical. Body dynamics were ignored and the residual conflict vector was weighted with a single free parameter. It was assumed that the sensory dynamics of the SCC were identical to those possessed internally by the central nervous system. They showed that for passive yaw rotation about the axis of gravity, if the central nervous system (CNS) compared actual SCC signals with those predicted using the internal model, the difference - weighted by a parameter (k_ω) - corresponded to the slow phase velocity of the VOR. The effect was to e.g. extend the duration of postrotatory eye movements well beyond that of the actual afferent firing duration, a phenomenon generally termed "velocity storage". Model predictions were consistent with animal VOR data and mathematically congruent with the models proposed by Robinson (1977) and Raphan and Cohen (1977, 1979). In fact, Merfeld was able to demonstrate that his observer structure was dynamically equivalent to these earlier single axis models. [1]

[1]Robinson's model utilized a low pass filter in a positive feedback loop. Raphan and Cohen employed a "velocity storage integrator" with a feed forward direct pathway. This model was later generalized to three dimensions (Raphan & Cohen 2002).

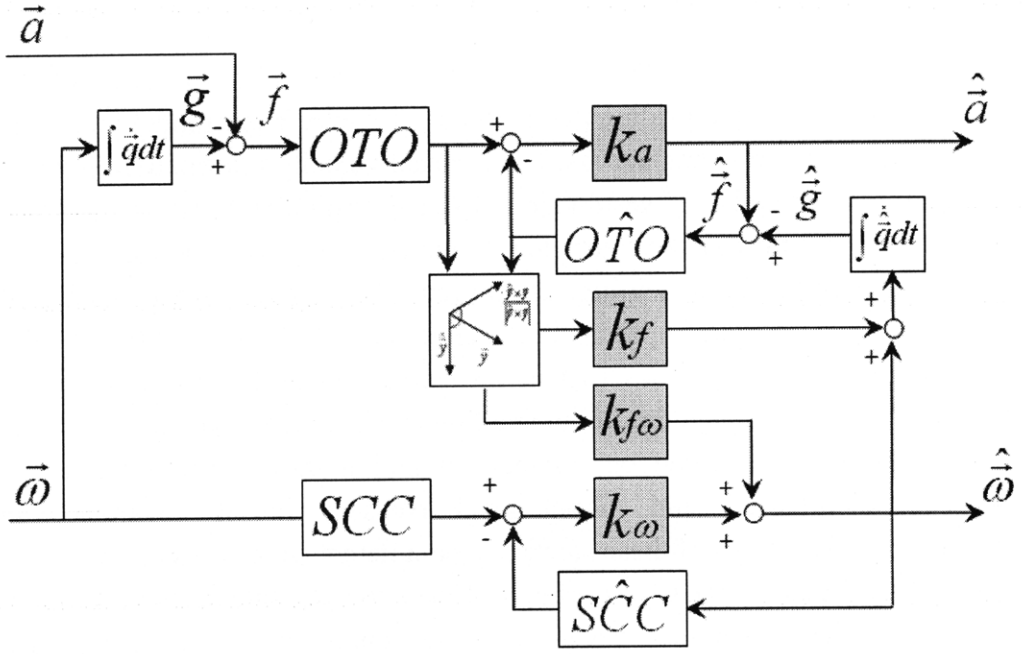


Figure 1. Merfeld 1993 Spatial Orientation Model. Three dimensional vectors of linear acceleration ($\vec{a}$) and angular velocity ($\vec{\omega}$) are input to the model in a head-fixed coordinate frame. Angular velocity is integrated using a quaternion integrator ($\int \dot{\vec{q}}\, dt$) to keep track of the orientation of gravity ($\vec{g}$) with respect to the head. The otolith (OTO) transfer functions are modeled as unity and respond to the gravito-inertial force (GIF) ($\vec{f} = \vec{g} - \vec{a}$). The semicircular canals (SCC) are modeled as a 2nd order high-pass filter with a cupula/endolymph long time constant of 5.7 sec and a neural adaptation time constant of 80 sec. Afferent signals from the canals and otoliths are compared in the central nervous system "observer" against expected values from a similar set of internal sensory dynamics ($\hat{SCC}$, $\hat{OTO}$). The resultant error signals are weighted with four free parameters (k_ω, $k_{f\omega}$, k_f, k_a) shaded in grey in the schematic above. The model outputs are central estimates of linear acceleration ($\hat{\vec{a}}$), gravity ($\hat{\vec{g}}$), and angular velocity ($\hat{\vec{\omega}}$).

Merfeld et al (1993) also extended their model to three dimensions, as shown in Figure 1. Tilt and acceleration perception and associated eye movements had been shown to not depend entirely on sense organ dynamics. Constant velocity eccentric rotation ("centrifugation") for example, elicits an illusory sensation of tilt that slowly aligns with the true gravitoinertial force vector. This gradual alignment process suggests that the CNS bases its estimate of gravity on both sensory output and expected gravitational information. Motivated by these observations Merfeld extended his orientation model to include additional internal estimates of gravity direction and magnitude ("gravity storage") and acceleration ("acceleration storage") (Figure 1). Canal and otolith dynamics were generalized to three dimensions and quaternion mathematics were used to represent the relationship between head and world coordinate frames. As shown in Figure 1, based on the internal model estimates of tilt and linear acceleration, the Observer predicts what otolith (OTO) and SCC afferent signals should be. The differences between actual and expected otolith signals were weighted by a parameter k_a to derive an acceleration estimate. The difference between the direction of actual and expected otolith cues, weighted by a

parameter k_f, was used to rotate the Observer's estimate of the direction of gravity towards reality. The rotation of the gravireceptor cue, weighted by a parameter $k_{f\omega}$, was used to adjust the Observer's estimate of angular velocity, acting in addition to the k_ω weighted SCC expectancy error, described earlier. By empirical adjustment of these four internal weighting parameters, this model was capable of predicting responses to a number to motion stimuli including constant velocity earth-vertical rotation, off-vertical-axis rotation (OVAR), and postrotational tilt.

TABLE 1. *Validation cases and weighting parameters for Observer and KF / EKF models*

	Stimuli Used for Validation						Parameter Values			
Observer Models	Earth Vertical Rotation	OVAR	Post-rotational Tilt	Fixed Cab Centrifugation	Linear Acceleration	Roll Tilt	K_a	K_f	$K_{f\omega}$	K_ω
Merfeld 1993	✓	✓	✓	-	-	-	-0.9	2	20	3
Haslwanter 2000[a]	-	✓	-	-	-	-	-1	10	1	1
Merfeld 2002	-	-	-	✓	-	✓	-2	2	2	3
Vingerhoets 2006[b]	-	✓	-	-	-	-	-4	2	8	8
Vingerhoets 2007[c]	-	✓	-	-	-	✓	-4	4	8	8
KF/EKF Models										
Borah 1979	✓	-	-	-	✓	-				
Pommellet 1990	✓	-	-	✓	✓	-		N/A		
Billien 1993	-	-	-	✓	-	-				

Extensions to Merfeld's 1993 Spatial Orientation Model. **(a)** Haslwanter et al (2000) implemented an alternate otolith model originally suggested by Dai et al (1989). Dai's model compensates for the inclined geometry of the utricular plane and was necessary to properly reproduce eye movements during large tilt angle ($\theta > 45°$) off-vertical axis rotation. A high pass filter was also added to the linear VOR pathway for additional modeling purposes. **(b)** Vingerhoets et al (2006) added a leaky integrator ($\tau = 0.04$) to the estimated linear acceleration pathway in order to obtain an estimate of translation velocity. This quantity was required for comparison with experimental data. **(c)** Vingerhoets et al (2007) included both the leaky integrator dynamics of the previous study and a weighted idiotropic vector to account for the static underestimation of the subjective visual vertical (SVV).

Model predictions were further validated and extended by Haslwanter et al (2000), Merfeld and Zupan (2002) and Vingerhoets et al (2006, 2007). Note that these models assumed that internal model estimates corresponded to perceptions, and that perceptions and eye movements were directly related. Table 1 summarizes these efforts and compares them against the alternate class of Kalman filter (KF) and extended Kalman filter (EKF) models. It is important to make a distinction between the validation methodologies employed for eye movement (e.g. Merfeld et al 1993, 2002, Haslwanter et al 2000) and perceptual models (e.g. Borah et al 1978, Vingerhoets et al 2007). Eye movement recordings provide the modeler with a complete time history of the slow phase eye velocity, and allow for a quantitative metric for data

fitting and parameter adjustment. Perceptual data is however limited to subjective descriptions of apparent sensations or tilt angles or durations as retrospectively described by subjects or snap-shotted at various points during the experiment. Perceptual model developers are then presented with the nontrivial task of reconstructing and fitting this limited "data" set as quantitatively best as they can. In considering both the table above and the results and simulations presented henceforth, it is important to keep this distinction and limitation in mind.

Note also that Observer models are deterministic. Unlike its Kalman Filter counterpart, Observer residual weighting values are left as free parameters of the model. They act as tuning knobs for the modeler and aid in making the model mimic actual data. Each parameter tunes one or more dynamic modes of model response. Through proper choice of these parameters, the Observer class orientation models were able to more reliably predict complex motion responses However, Observer models developed so far have been limited to SCC and otolith cue interaction. The model presented in the next chapter builds out using the core of Merfeld's vestibular-only model to include additional state estimates and both static and dynamic visual cue inputs. The extended model retains the vestibular core of the Observer model implemented by Merfeld, but estimates azimuth as well as tilt. Previous models predicted orientation and linear acceleration, but did not predict position in space. To do this, an additional ("limbic") coordinate frame, aligned with the perceived vertical, was added, and velocity and position "path" integration was assumed to take place in this fame. The vestibular-only portion of the extended model was then tested, and found to reproduce results for stimulus paradigms listed in Table 1 and described in papers by Haslwanter et al (2000), Merfeld & Zupan (2002), and Vingerhoets et al (2006,2007). As described in the next chapter, visual pathways were then added and extended model results were compared with the KF model results and original data considered by Borah for the simple visual-vestibular motion paradigms of linear vection, circular vection, rotation in the light and acceleration in the light. With this validated parameter set in place, Coriolis, pseudo-Coriolis, Tilt-Gain and Tilt-Translation illusions were modeled and simulated.

CHAPTER 3. MODEL DEVELOPMENT

3.1 Vestibular Model

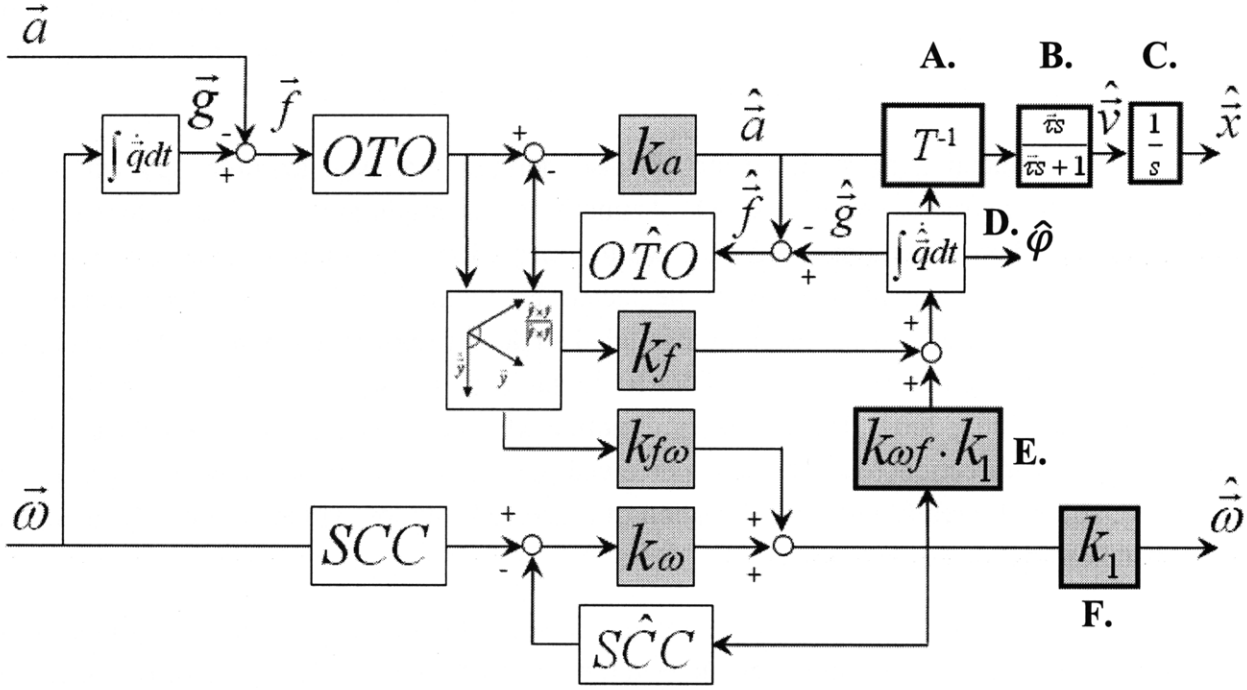


Figure 2. Extended Vestibular Model. Modifications to the original Merfeld & Zupan (2002) model are outlined in black and denoted **A - F**. **(A)** Head to limbic coordinate frame transformation. **(B)** Leaky integrator for velocity estimate ($\hat{\vec{v}}$). Note Merfeld & Zupan included a similar leaky integrator to obtain velocity estimates for the translational component of the VOR. **(C)** Integrator for position estimate ($\hat{\vec{x}}$). **(D)** Estimated azimuth **(E-F)** Additional feedback gains $k_{\omega f}$ and k_1. The free parameters, shaded in grey, have been set according to the values shown in Table 2 (Section 4.1). ($\vec{x}$ -position, $\vec{v}$ -velocity, $\vec{a}$ -acceleration, $\vec{g}$ -gravity, $\vec{f}$ -GIF, $\vec{\omega}$ - angular velocity, $\vec{\tau}$ –leaky integration time constant, φ -azimuth; $\hat{}$ denotes estimated quantity, e.g. $\hat{\vec{x}}$ -estimated position.)

The vestibular core of the Observer visual-vestibular interaction model (Figure 2) is a modified and extended version of the model proposed by Merfeld and Zupan (2002). The topology of the model as shown in the figure has been rearranged so it resembles the presentation format of Haslwanter et al (2001) and extended to include the additional state estimates (position $\hat{\vec{x}}$ and velocity $\hat{\vec{v}}$) necessary for displacement estimation and visual sensory interaction. To obtain these estimates, we assume that the CNS integrates the perceived linear acceleration vector ($\hat{\vec{a}}$) in an allocentric reference frame oriented to the local vertical.[2]

---

[2] We refer to this frame, which is aligned with the perceived gravitational horizontal as the "limbic" coordinate frame, since a variety of physiological evidence suggests that estimates of azimuth and direction originate in limbic areas of the brain, including hippocampus, thalamus and medial entorhinal cortex. Neural coding of place and grid cells (Best et al 2001, Hafting et al 2005, Calton and Taube 2005, Knierim et al 2000, 2003, Oman 2007), along with orientation and way finding experiments performed in 1- (Aoki et al 2003, 2005) and 0-G (Vidal et al 2003, 2004), suggest that "innate neurocognitive functions appear to be specialized for natural 2D navigation about a gravitationally upright body axis." (Vidal et al 2004)

This "limbic coordinate frame" is defined by the quaternion vector ($\hat{\vec{q}}$) from the estimated gravitation state ($\hat{\vec{g}}$). At each time step of the simulation the estimated linear acceleration vector is transformed to the limbic coordinate system and integrated twice to obtain estimates of position and velocity. For a detailed description of the quaternion mathematics and transformation methods employed, the reader is referred to Appendix A.

The estimated quaternion vector ($\hat{\vec{q}}$) also defines perceived azimuth ($\hat{\varphi}$) within the limbic coordinate frame (Appendix A). Azimuth is an important physical estimate with implications in both laboratory and real world orientation/navigation simulations. Previous Observer model implementations have neglected an azimuth calculation. One interesting prediction is the sustained progression of azimuthal angle in response to off vertical axis rotation (OVAR). The classic description of an OVAR simulation, as described by Denise et al (1988) and Wood et al (2002, 2007), is a sensation of motion which proceeds along a conical path while facing the same direction (e.g. constant perceived azimuth). The model, however, predicts a sustained, although greatly reduced, sensation of perceived rotation which continually alters the estimated heading, or azimuth angle.

The integration of acceleration to velocity is accomplished with a leaky integrator[3] with individual time constants for motion about each limbic coordinate axes ($\vec{\tau} = [16.66, 16.66, 1.0]^T$). The values of time constants listed capture the qualitative characteristics of large physical scale integrated self motion.

It should be noted that time constants for motion within the horizontal plane differ substantially from those associated with vertical motion along the actual or perceived direction of gravity. Position and velocity estimation within the horizontal plane has been shown to be fairly accurate for a range of motion amplitudes and frequencies (Israel and Berthoz 1989, Mittelstaedt et al 2001, Seidman 2008, Guedry and Harris 1963, Mittelstaedt and Glasauer 1991, Loomis and Klatzky 1993). Studies performed in helicopters and vertical motion simulators (Walsh 1964, Malcolm & Jones 1973, Jones et al 1978) have shown a fundamental difference in the ability of humans to integrate inertial acceleration cues along a gravitationally vertical axis. Experimental subjects were found unable to correctly indicate the magnitude or direction of motion, often eliciting phase errors of 180 degrees. While aware of vertical displacement, subjects could indicate the proper direction of travel only slightly better than chance. To model these large phase and magnitude estimation errors the leaky integration time constant about the perceived vertical has been substantially shortened (Figure 3).

[3] A standard integrator was implemented for the velocity to position integration in order to ensure that a static visual position input would result in a dynamic response with zero steady state error. Leaky dynamics initiate phase and magnitude estimation errors which do not correspond to perceived reality (e.g. displacement estimates for sinusoidal horizontal translatory motion should remain veridical with a visual position reference).

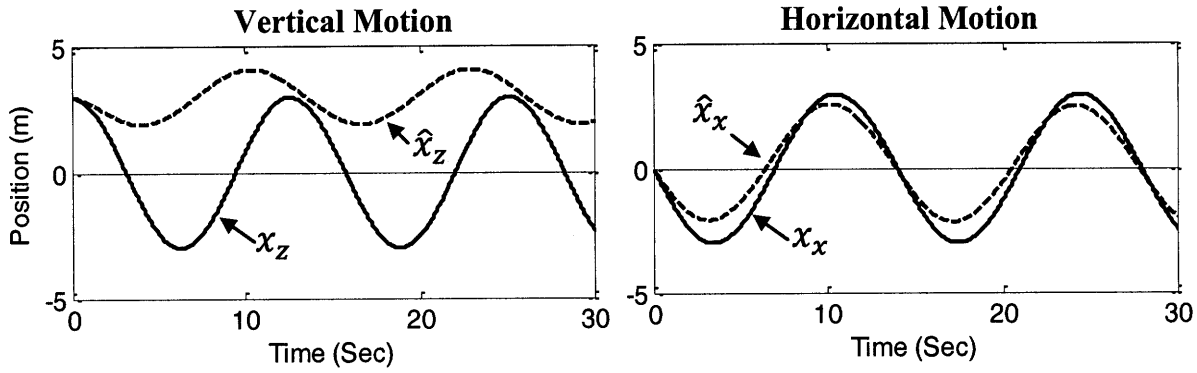


Figure 3. Model response to sinusoidal vertical and horizontal displacement profiles. Horizontal motion is perceived fairly accurately while vertical motion exhibits large phase and magnitude errors.

Two additional residual weighting parameters have also been added to the model. The first parameter ($k_{\omega f}$) is nominally set to 1.0 and allows for user control over the influence of angular velocity on the rate of change of gravity. This parameter was added for the simulation of Tilt-Gain and Tilt-Translation illusions (Section 4.7). The second parameter (k_1) is a function of the angular velocity residual weighting parameter ($k_1 = (k_\omega + 1)/k_\omega$) and was required to make the loop gain of the angular velocity feedback loop unity.[4]

[4] The loop gain is based on the angular velocity weighting parameter (k_ω) and is calculated as $k_\omega/(k_\omega + 1)$. Using the Merfeld et al (1993, 2002) parameter ($k_\omega = 3$) the loop gain is found to be 0.75. This gain was intentionally set to mimic the 70% angular VOR response for eye movement data yet is inconsistent with perceptual responses for many simple experiments (i.e. static tilt, constant velocity yaw rotation), where the initial response to sudden head movements is veridical.

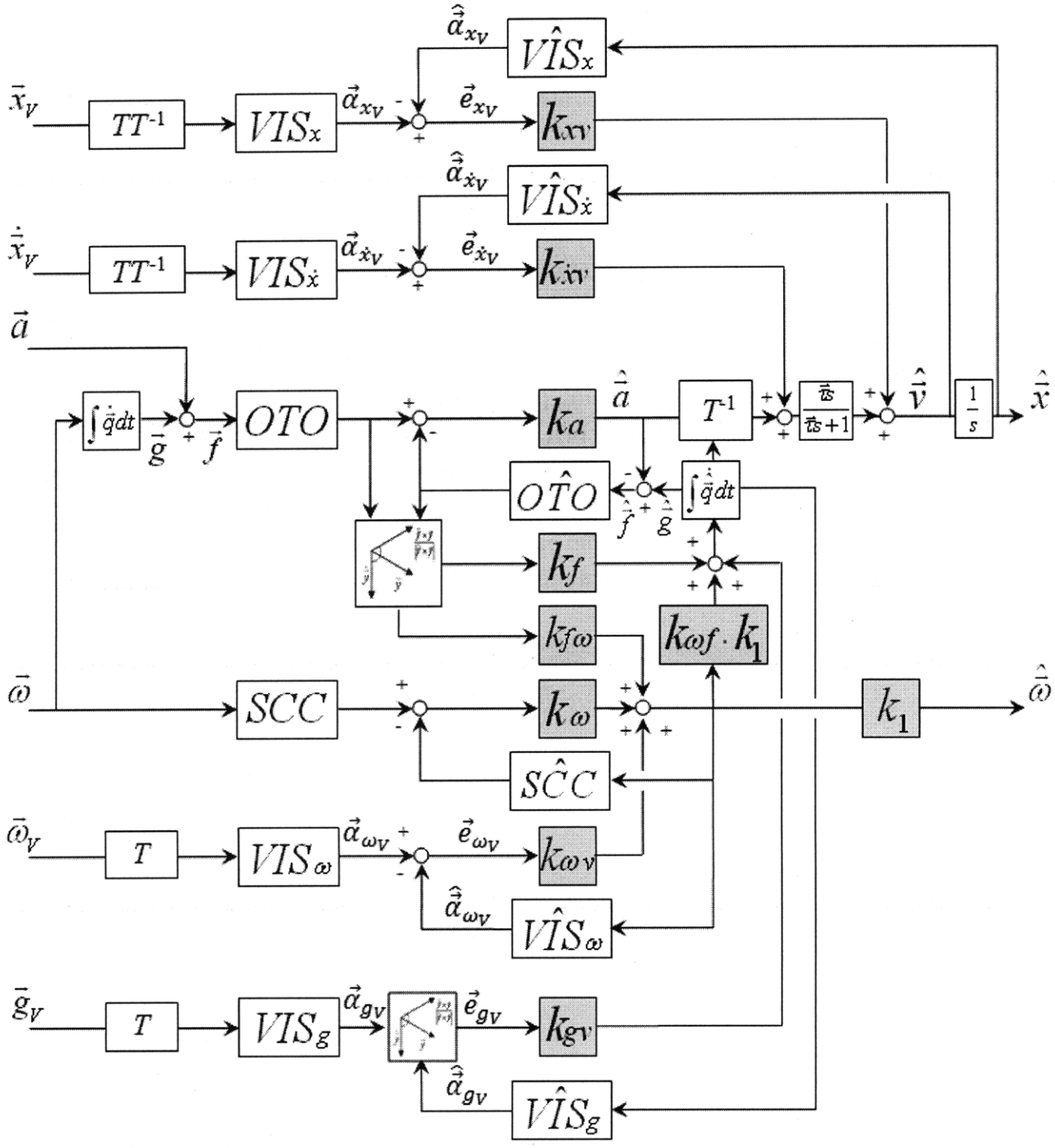


Figure 4. Visual - vestibular interaction model. Static and dynamic visual inputs are added to the extended vestibular model of Section 3.1. Model inputs now include static visual position ($\vec{x}_V$) and gravity ($\vec{g}_V$) and dynamic visual velocity ($\dot{\vec{x}}_V$) and angular velocity ($\vec{\omega}_V$). All cues are centrally combined and used to generate internal estimates of angular velocity ($\hat{\vec{\omega}}$), acceleration ($\hat{\vec{a}}$), velocity ($\hat{\vec{v}}$), position ($\hat{\vec{x}}$) and gravity ($\hat{\vec{g}}$). Vestibular and visual free parameters are highlighted in grey. Values for the free parameters are shown in Section 4.1.

3.2 Visual – Vestibular Interaction Model

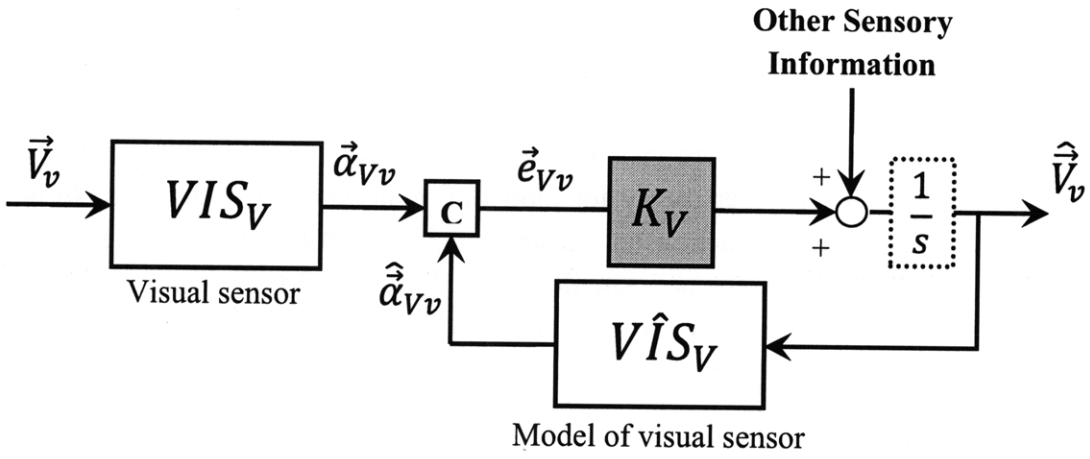


Figure 5. Block diagram representation for a generic visual model pathway. *See text for explanation.*

The preceding vestibular model extensions described in Section 3.1 were essential prerequisites for the addition of visual sensory information. With this modified vestibular model in place, a visual-vestibular sensory interaction model is now proposed (Figure 4).

The general case

To keep the model structure and notation consistent with the original Merfeld et al (1993) model, each visual pathway is constructed as shown in Figure 5. For a generic visual pathway *V*; a visual input ($\vec{V}_v$) is processed by the visual sensor (VIS_V) to generate a visual sensory estimate ($\vec{\alpha}_{Vv}$). This estimate is compared (**C**) to an expected visual sensory estimate ($\hat{\vec{\alpha}}_{Vv}$) from an internal model of the visual sensor ($V\hat{I}S_V$). The comparative difference ($\vec{e}_{Vv}$) ("sensory conflict") is weighted with a residual weighting parameter (K_V) and added to the rate of change of the estimated state ($\hat{\vec{V}}_v$). The weighted conflict vector is added to the derivative of the state in order to keep the visual model additions consistent with the structure of a classic Luenberger Observer. Since Merfeld did not include an integrator in the forward loop of the angular velocity feedback pathway, we add the weighted visual angular velocity error directly to the state itself.

Physical variables and coordinate frames

We assume that the visual system is capable of extracting four visual cues from its environment. These are position ($\vec{x}_V$), velocity ($\dot{\vec{x}}_V$), angular velocity ($\vec{\omega}_V$), and gravity/"down" ($\vec{g}_V$). The visual input variables are represented by three – dimensional vectors in a right handed, orthogonal, world-fixed, frame of reference (X_W, Y_W, Z_W). To ensure congruency with the Observer model's head and limbic coordinate frames, the visual cues are transformed to their respective frames of interaction prior to sensory processing. Visual gravity and angular velocity are transformed to the head-fixed coordinate axes and visual position and velocity are transformed to the perceived limbic frame (Appendix A).

Sensor dynamics

For simplicity we assume that the visual system sensory dynamics can be approximated as unity for both static and dynamic visual inputs. Unlike Borah and Pommellet, we do not distinguish between focal and ambient vision or account for visual saturation limits for linear and circular vection cues. This over-simplified visual model allows for a baseline assessment of the usefulness and practicality of Observer theory for modeling multi sensory interaction. In three dimensional space we can represent each visual sensor as a 3x3 identity matrix. Since dynamic inputs illicit a sensation of motion in the opposite direction of the visual field (e.g. linear vection and circular vection), the dynamic sensors are modeled as negative 3x3 Identity matrices. Each sensor transforms visual input ($\vec{x}_V\ \dot{\vec{x}}_V\ \vec{\omega}_V\ \vec{g}_V$) to visual sensory estimates ($\vec{\alpha}_{x_V}\ \vec{\alpha}_{\dot{x}_V}\ \vec{\alpha}_{g_V}\ \vec{\alpha}_{\omega_V}$).

Static Visual Cues

$$\frac{\vec{\alpha}_{x_V}}{\vec{x}_V} = VIS_x = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3} \qquad \frac{\vec{\alpha}_{g_V}}{\vec{g}_V} = VIS_g = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3}$$

Dynamic Visual Cues

$$\frac{\vec{\alpha}_{\dot{x}_V}}{\dot{\vec{x}}_V} = VIS_{\dot{x}} = \begin{bmatrix}-1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & -1\end{bmatrix} = -I_{3\times3} \qquad \frac{\vec{\alpha}_{\omega_V}}{\vec{\omega}_V} = VIS_{\omega} = \begin{bmatrix}-1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & -1\end{bmatrix} = -I_{3\times3}$$

Internal sensors dynamics

We assume that the CNS possesses accurate internal models for each visual sensor. Since the CNS already accounts for the proper direction of the visual estimate, we can represent all internal models of visual sensory dynamics as positive 3x3 identity matrices. The internal model of the visual sensors transform the central state estimates ($\hat{\vec{x}}_V\ \hat{\dot{\vec{x}}}_V\ \hat{\vec{g}}_V\ \hat{\vec{\omega}}_V$) to expected visual sensory estimates ($\hat{\vec{\alpha}}_{x_V}\ \hat{\vec{\alpha}}_{\dot{x}_V}\ \hat{\vec{\alpha}}_{g_V}\ \hat{\vec{\alpha}}_{\omega_V}$).

$$\frac{\hat{\vec{\alpha}}_{x_V}}{\hat{\vec{x}}_V} = V\hat{I}S_x = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3} \qquad \frac{\hat{\vec{\alpha}}_{g_V}}{\hat{\vec{g}}_V} = V\hat{I}S_g = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3}$$

$$\frac{\hat{\vec{\alpha}}_{\dot{x}_V}}{\hat{\dot{\vec{x}}}_V} = V\hat{I}S_{\dot{x}} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3} \qquad \frac{\hat{\vec{\alpha}}_{\omega_V}}{\hat{\vec{\omega}}_V} = V\hat{I}S_{\omega} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = I_{3\times3}$$

Error Calculations

A sensory conflict vector is calculated for each visual input based on the relative error between the actual and expected visual sensory estimates. The visual position, velocity and angular velocity errors are calculated through vector subtraction. Each error is represented as a vector containing an individual sensory conflict for each orthogonal axis.

$$\vec{e}_{x_V} = \vec{\alpha}_{x_V} - \hat{\vec{\alpha}}_{x_V}$$

$$\vec{e}_{\dot{x}_V} = \vec{\alpha}_{\dot{x}_V} - \hat{\vec{\alpha}}_{\dot{x}_V}$$

$$\vec{e}_{\omega_V} = \vec{\alpha}_{\omega_V} - \hat{\vec{\alpha}}_{\omega_V}$$

The gravitational error requires both a magnitude and directional component. We calculate the conflict vector between the actual and expected gravitational sensory estimates by computing the rotation required to align both vectors. For the directional component, we use a cross product to calculate a unit vector perpendicular to the plane formed by the two vectors.

$$\frac{\vec{e}_{g_V}}{|\vec{e}_{g_V}|} = \frac{\vec{\alpha}_{g_V} \times \hat{\vec{\alpha}}_{g_V}}{|\vec{\alpha}_{g_V} \times \hat{\vec{\alpha}}_{g_V}|}$$

For the magnitude, we use a dot products to calculate the angle required to align both vectors within the previously calculated plane. Note – this implementation is identical to Merfeld's GIF rotational error.

$$|\vec{e}_{g_V}| = \cos^{-1}\left(\frac{\vec{\alpha}_{g_V}}{|\vec{\alpha}_{g_V}|} \cdot \frac{\hat{\vec{\alpha}}_{g_V}}{|\hat{\vec{\alpha}}_{g_V}|}\right)$$

Residual Weighting Parameters

The error signals are individually weighted with residual weighting parameters that can be adjusted by the modeler to fit data. The visual gravity residual weighting parameter (K_{gv}) determines the influence of the visual gravitational error ($\vec{e}_{g_V}$) on the rate of change of the internal estimate of gravity ($\hat{\vec{g}}$). The visual angular velocity residual weighting parameter ($K_{\omega v}$) determines the influence of the visual angular velocity error ($\vec{e}_{\omega_V}$) on the internal estimate of angular velocity ($\hat{\vec{\omega}}$). The visual position residual weighting parameter (K_{xv}) determines the influence of the visual position error ($\vec{e}_{g_V}$) on the rate of change of the internal estimate of position ($\hat{\vec{x}}$). The visual velocity residual weighting parameter ($K_{\dot{x}v}$) determines the influence of the visual velocity error ($\vec{e}_{\dot{x}_V}$) on the rate of change of the internal estimate of velocity ($\hat{\vec{v}}$). The values and methodologies used to set the weighting parameters are detailed in Section 4.1.

3.3 OBSERVER Spatial Orientation Analysis Tool

Figure 6. OBSERVER Spatial Orientation Analysis Tool main GUI.

The OBSERVER Spatial Orientation Analysis Tool is a MATLAB/Simulink based program developed to aid in the processing, simulation and visualization of human perception in response to 3D, complex, multisensory motion stimuli. OBSERVER is designed to be more easily used by sensorimotor investigators, human factors engineers and by disorientation incident/accident investigators.

The OBSERVER graphical user interface (Figure 6) provides users with complete control over all model parameters and data input/output streams without a need for advanced programming skills or linear systems knowledge. The GUI communicates with a MATLAB subroutine file that passes data to and from the visual-vestibular Observer block diagram model in Simulink. Input time series data for both visual and vestibular information is supplied via Excel spreadsheet, using a specified format. Though originally validated for 1-G data on Earth, users can select alternate gravity environments for simulations in Lunar (1/6G), Martian (3/8G) or 0-G. All of the previous mentioned residual weighting parameters, leak rates, and sense organ time constants are directly editable on the main GUI. The default OBSERVER parameters correspond to those used for all simulations presented throughout this thesis. Once a file is loaded, configured, and simulated the user can export or visualize the resultant perceptual data with a family of built in visualization tools (Figure 7).

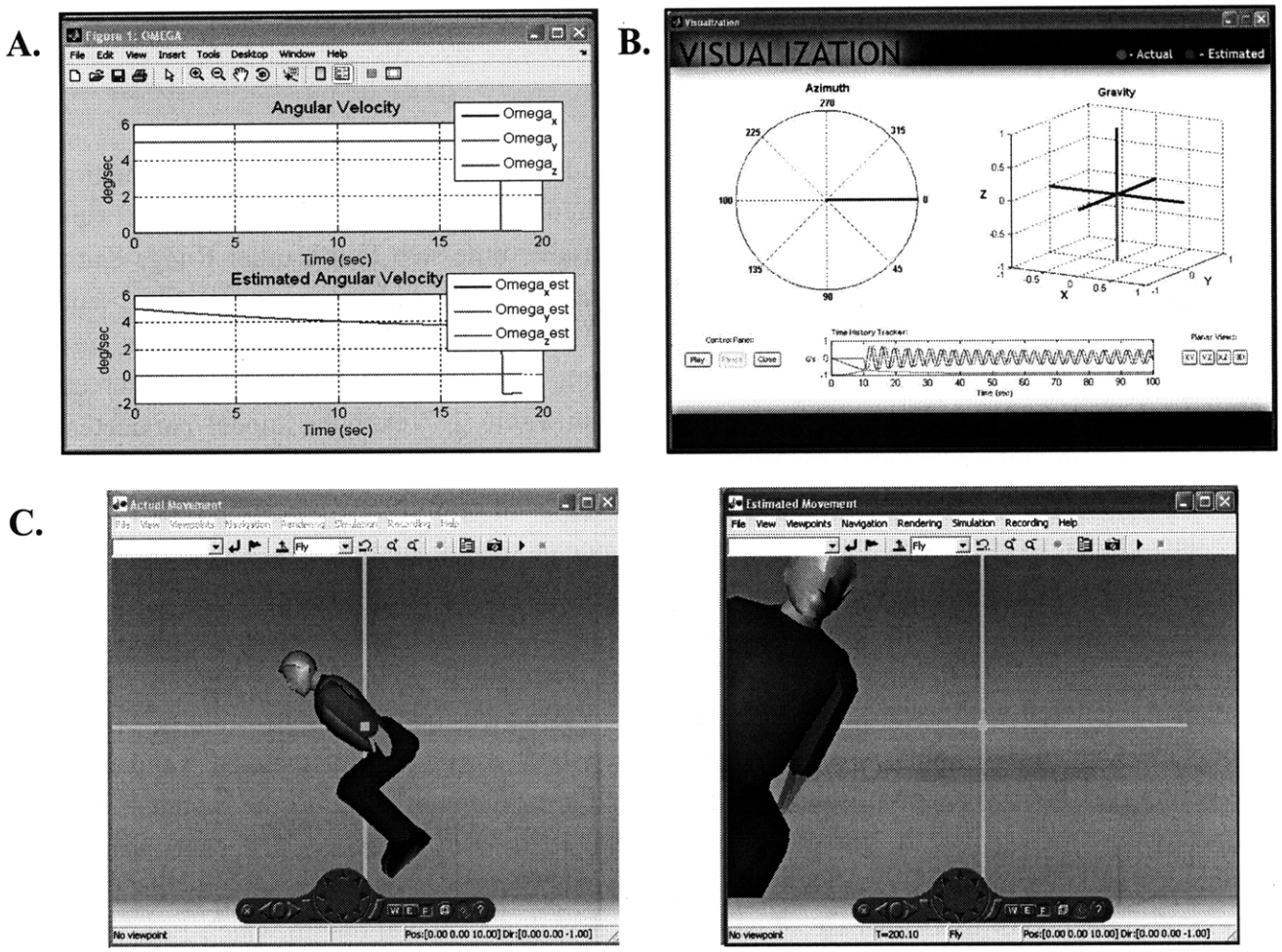


Figure 7. (A) Standard OBSERVER output data plot window. Plot displays the actual and estimated response for each individual vector component of a particular model output. OBSERVER provides 9 default plots; Gravity, GIF, Linear Acceleration, Linear Velocity, Position, Angular Velocity, Tilt/Subjective Visual Vertical (SVV), Euler Angles, and Stimulus Cues. **(B)** 3D animated vector plot of the actual and perceived direction of gravity. Users can view the vector plot in the standard 3D isometric view and also with respect to each of the head axis planes. An animated progression of actual and predicted azimuth is also presented. **(C)** Virtual reality simulation of the actual and estimated motion response. The VR simulation allows for a side by side comparison of the rotational and translational response of the subject in a true world fixed coordinate frame.

Each estimated model output can be plotted and compared with the actual response in a standard MATLAB plot window, (Figure 7A). This allows users the ability to use the more advanced MATLAB plotting features for further data regression and analysis. A separate 3D visualization window dynamically displays the time course of observer model "down" and "azimuth" estimates, (Figure 7B). Finally, users can view the motion response in 3D with a virtual reality simulation of side by side virtual manikins representing the actual and estimated motion. A complete walkthrough of these features and additional OBSERVER information is provided in the OBSERVER Spatial Orientation Analysis Tool User Guide (Appendix B).

CHAPTER 4. RESULTS AND DISCUSSION

4.1 Simulation

The Observer visual-vestibular interaction model at the core of the spatial orientation analysis tool was implemented in the MATLAB – Simulink 2007b (*The MathWorks*) software suite. The Simulink model was configured with a variable time-step fourth order Runge-Kutta differential equation solver (*ode45 Dormand-Prince).* All simulations, input files, and visualization output were processed with the OBSERVER spatial orientation analysis tool (Appendix B). The Observer model residual weighting parameters were left constant for all simulations and set according to the values shown in Table 2. Vestibular model parameters match those originally calculated by Vingerhoets (2007). Vingerhoets' parameters reflect the only Observer model vestibular weighting scheme validated entirely on perceptual data[5], and were thus chosen over the eye movement based parameter sets of Merfeld et al 1993, 2002 and Haslwanter et al 2000. One should note that the vestibular model extensions described in *Section 3.1* were not accounted for in the Vingerhoets' validation. For the vestibular-only simulations presented in this section, these model differences were negligible. The additional visual parameters, denoted with a subscript "*v*", were tuned to match the modeling results of Borah, Young and Curry (1978) and the subsequent data sets they considered for their KF validation. The tuning process was accomplished with a trial and error method based on the pertinent data characteristics responsive to parameter adjustment (e.g. rise times, steady state values, amplitudes, and phase angles). For cases where the Borah model was fit to experimental data, the simulated curves were used for ease of comparison. For those simulations in which Borah and colleagues could not represent pertinent data characteristics (e.g. vection, where the simulated velocity rise time was significantly longer than the value reported by Chu (1976)), the data itself was used.

TABLE 2. *Residual Weighting Parameters*

	Vestibular Parameters					Visual Parameters				Leaky Time Constants		
	K_a	K_f	$K_{f\omega}$	K_ω	$K_{\omega f}$	K_{x_v}	$K_{\dot{x}_v}$	K_{g_v}	K_{ω_v}	τ_x	τ_y	τ_z
Value	-4	4	8	8	1	0.1	0.75	10	10	16.67	16.67	1

Sections 4.2 – 4.4 present the Observer model's results for the basic visual and visual – vestibular motion paradigms simulated by Borah (1978). These cases were considered fundamental to the validation of the model's visual system dynamics and were used for the above mentioned tuning of the visual residual weighting parameters.

[5] Through a limited parameter space search method these gains were found to minimize the sum of squares error ("SSE") for translation precepts during OVAR.

Sections 4.5 – 4.7 present modeling results for several vestibular and visual – vestibular illusions including Coriolis, pseudo Coriolis and the Tilt-Gain and Tilt-Translation illusions. These cases were chosen to illustrate the predictive capability of the Observer model for more complex multisensory motion profiles.

4.2 Constant Velocity Rotation about an Earth Vertical Axis

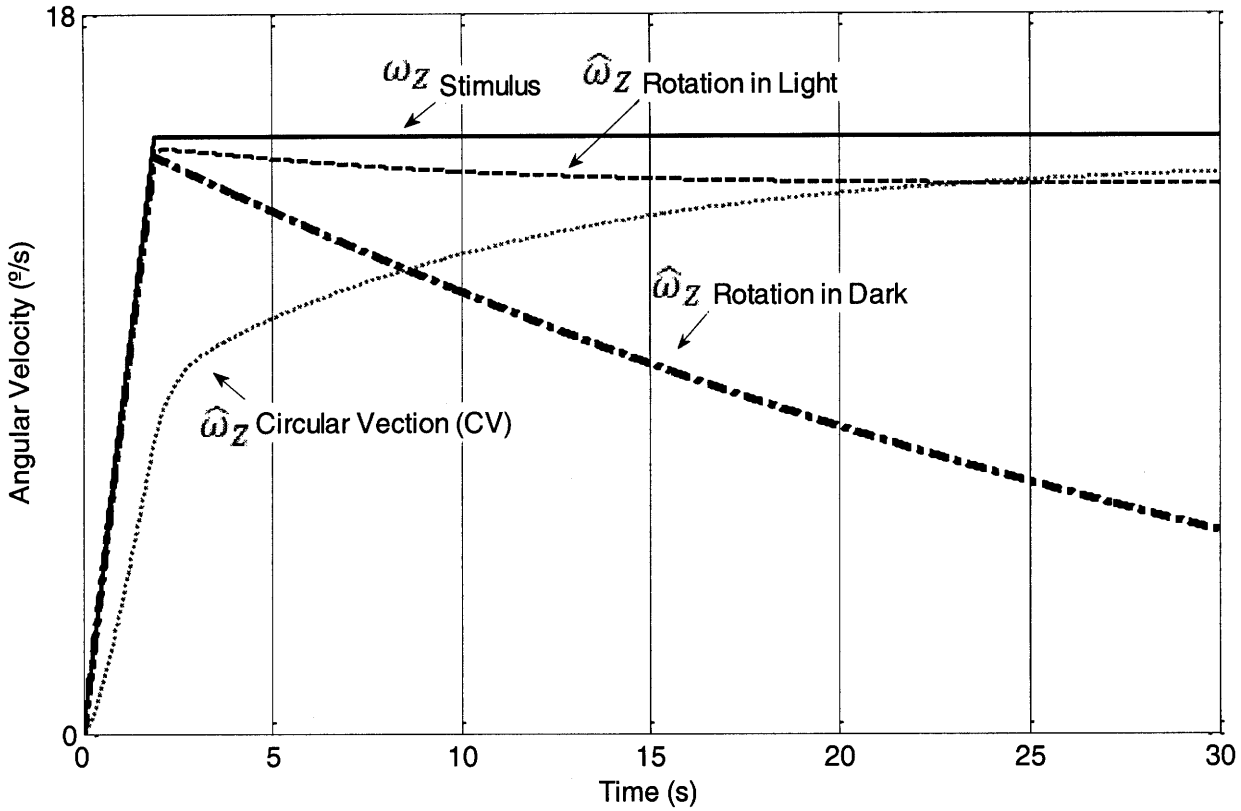


Figure 8. Model predictions for constant velocity rotation about an Earth vertical axis. **Rotation in Light/Dark**: The simulated subject was seated upright in a rotary chair and rotated in the light/dark at ω_Z= 0.26 radian/s (14.89°/s) for a duration of 30 seconds. **Circular Vection**: The subject is stationary and placed inside an optokinetic drum which rotates in the light at ω_Z= -0.26 radian/s (14.89°/s). Drum rotation is in the opposite direction of the angular velocity stimulus to illicit an illusory sensation of rotation which has a consistent direction with the angular motion in the Light/Dark example. The Circular Vection and Rotation in the light response mimic the modeling results of Borah (1978) and the experimental data of Waespe and Henn (1977).

Figure 8 shows the model response for constant velocity rotation about an Earth vertical axis under three different experimental conditions: rotation in the light, rotation in the dark, and rotation of only the visual surround (circular vection). Velocity perception in the dark (curve labeled Rotation in Dark) displays the typical exponential decay of angular sensation and the lengthened time constant attributed to central velocity storage. Since visual information is not present in this condition, the simulation results are similar to Merfeld's 1-D velocity storage model (1993). When a stationary visual surround is present (curve labeled Rotation in Light) the

model predicts a sustained sensation of rotational motion which slightly decays to a value close to the actual input stimuli.

Finally, the model successfully predicts an illusory sensation of angular velocity when subjected to pure rotation of the surrounding visual field (curve labeled Circular Vection). The circular vection response curve shows two distinct components associated with the time course of the perceived self motion; a fast rising component responsible for the quick initial onset followed by a slow rising component which levels out to a value slightly below the velocity of the visual field. The fast rising component accounts for 70% of the total angular velocity estimate (0°/s →10.3°/s in 2.5 seconds) and is driven by the visual system and the dynamics of the visual velocity residual feedback loop. As the internal model's estimate of angular velocity increases, visual-vestibular interactions begin to slow down the neurological processing of rotational motion. These more gradual dynamics result from the velocity storage time constant associated with the canals and CNS and account for the remaining 30% of velocity perception (10.3°/s → 14.5°/s in 37.75 seconds). The existence of these separate components has been demonstrated experimentally (Cohen et al 1981, Jell et al 1984, Lafortune et al 1986) and was not predicted by either the Borah KF model or the Pommellet EKF model.

Although the overall circular vection simulation is good, like the Borah and Pommellet models, the Observer model predicts an immediate onset of circular vection sensation at the start of visual field motion. This is inconsistent with the delay typically observed in human subjects. (To account for vection delays and inter-subject variability Borah implemented a non-linear ad-hoc conflict mechanism which could distinguish and schedule gains for pure rotational field motion. Without a similar ad-hoc augmentation, our Observer model is incapable of replicating this vection onset latency.)

4.3 Forward Linear Acceleration on a Sled (Somatogravic Illusion)

Results (Figure 9A, B) show that the model successfully predicts the somatogravic ("pitch up") illusion for forward linear acceleration in the dark and the suppression of such illusion in the light. The somatogravic illusion in the dark has been well documented experimentally (Cohen et al 1973, Graybiel et al 1951, Graybiel 1966) and is predicted in both Borah's and Pommellet's KF and EKF models. It is worth comparing the model predictions for the lighted condition, where the visual portions of the Observer model work differently than the other two models. The Observer model uses visual angular velocity and visual gravity information, along with canal and otolith cues from the vestibular system, to estimate acceleration and pitch angle. While visual linear velocity and position information are also available, the structure of the Observer model posits that these quantities cannot affect the lower integrals of acceleration or the relative orientation of the simulated subject. This is quite different than the KF and EKF modeling implementations. Borah and Pommellet's models did not incorporate visual cues to the direction of gravity, and assumed that only visual velocity information, resultant from linear motion of the moving scene, would suppress the somatogravic

illusion and produce correct estimates of forward linear acceleration. In a study addressing this specific issue, Tokumaru et al (1998) found that subjects who underwent linear acceleration with isolated visual vection cues without visual gravity cues reported sensations of tilt with equal likelihood and magnitude as those without any visual information at all. They additionally found that when subjects were provided with a visual gravity reference they felt significantly reduced magnitudes of the pitch up sensation. These findings suggest that while all three models properly predict visual suppression of the somatogravic illusion, the Observer model's implementation is more consistent with the actual visual-vestibular interaction mechanism responsible.

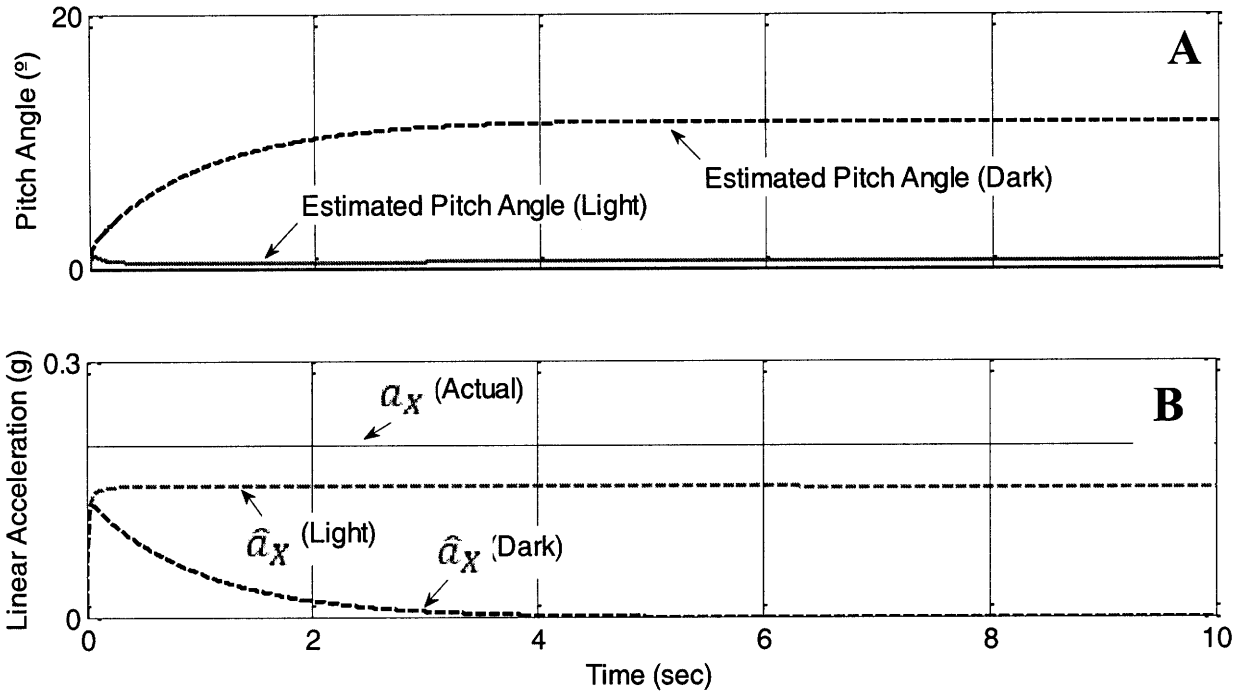


Figure 9. Model response to a step in forward linear acceleration. The simulated subject is seated upright and accelerated forward (+X) on a horizontal sled at 0.2g's for 10 seconds in both darkness and the lighted conditions. The time course and dynamics of the predicted pitch up sensation were set to match the experimental centrifuge data from Graybiel (1951) and the Borah KF response curves (1978). **(A)**: Estimated pitch angle for darkness and lighted conditions. **(B)**: Estimated linear acceleration ($\hat{a}_X$) for darkness and lighted conditions. Also shown is the 0.2g input stimulus (a_X).

4.4 Forward Linear Vection

In this simulation the subject is seated upright and views a high fidelity moving visual scene. Input to the model is represented as a -15 cm/s step in visual linear velocity ($\dot{\hat{x}}_V$). As the CNS gradually accepts the visual input, an illusory sensation of linear motion ("linear vection") develops in a direction opposite to visual field motion (Figure 10). (In the Figure 10 simulation, the leaky acceleration to velocity integrator was set to $\vec{\tau} = \infty$. In this case the simulation predicts a steady state velocity estimate equal in magnitude to the velocity of the visual field input ($\hat{\dot{x}}_X$ = 15cm/s). Zero steady state perceptual error then results from the perfect integrator in the forward loop of the velocity feedback pathway. Presumably the subject perceives the visual scene to be

stationary. For the nominal value of the leak rate (Table 2), the steady state perceived velocity gain will be 0.926. Presumably this means that the subject will perceive that the visual scene is moving in the opposite direction.

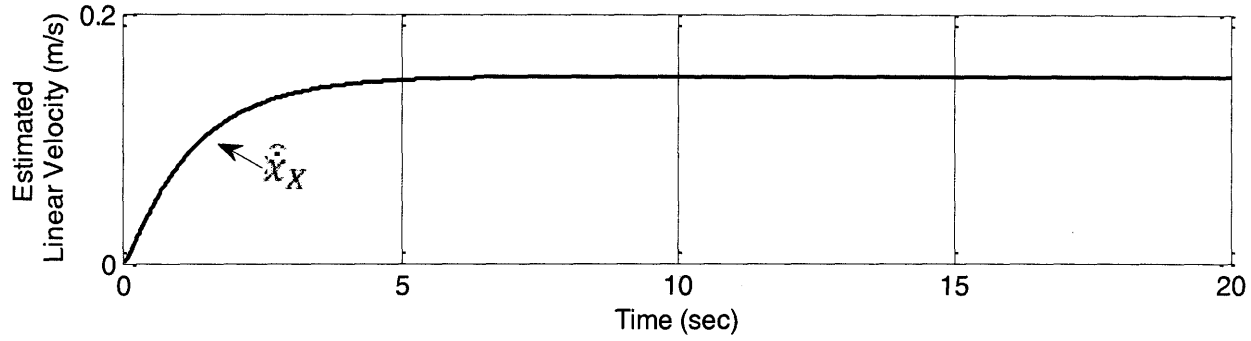


Figure 10. Predicted model response to a -15 cm/s linear vection stimulus.

Model predictions were set to match the results of Borah and Pommellet's vection simulations as well as the experimental vection translatory velocity data of Berthoz et al (1975) and Chu (1976). As with circular vection, all three models fail to predict vection onset delays.

Finally, notice also that due to the structure of the visual residual pathways assumed in the Observer model, changes in visual flow velocity are not interpreted as acceleration, and do not influence the estimated direction of gravity. This is why in this example, a step change in visual field velocity does not result in any transient tilt.

4.5 Vestibular "Coriolis" Cross-coupling

During this simulation the subject sits head erect in a chair that is rotated counter-clockwise in yaw about an Earth vertical axis at ω_Z = 57.3°/s (1 rad/s) for 120 seconds in the dark. At 60 seconds, when angular velocity perception has effectively decayed to zero, the simulated subject makes a rolling head tilt of +30° (60°/s for 0.5 seconds) towards the right shoulder. The resultant stimuli to the superior and horizontal semicircular canals produces a response traditionally referred to as the vestibular Coriolis Effect. The result is an illusory sensation of angular motion and tilt about a third axis of rotation which can be highly nauseogenic when a gravity cue is simultaneously present.

These simulation results (Figure 12) are in good agreement with the vectoral analysis of Guedry and Benson (1978). (For an identical simulation their vector analysis predicted an estimated angular velocity magnitude of $\hat{\vec{\omega}}$ = 0.502 rad/sec with individual components of $\hat{\omega}_Y$= 0.5 rad/sec and $\hat{\omega}_Z$= -0.13 rad/sec. This corresponds to an illusory pitch down sensation about an axis 74.6° from the true vertical. The small differences between our modeling results and Guedry's theoretical calculations can be attributed to the 0.5 second latency in the rolling head tilt. Guedry and Benson assumed that the tilt was accomplished instantaneously).

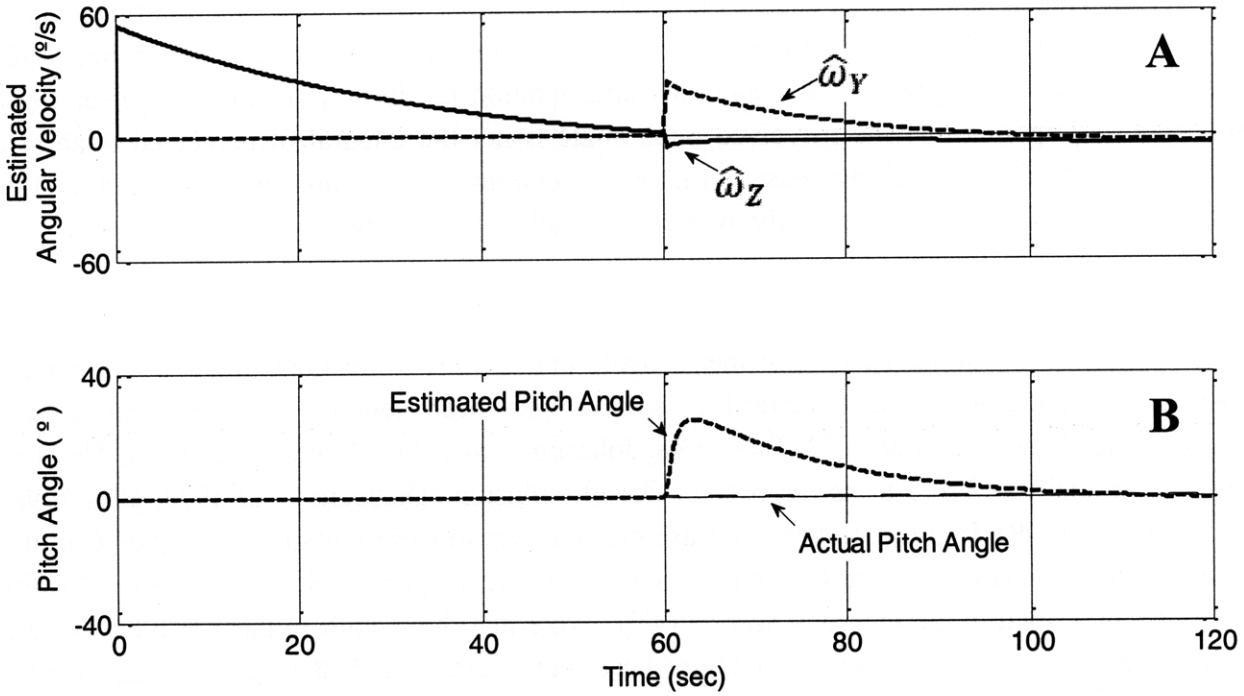


Figure 11. Simulation of vestibular Coriolis effect. The head is rolled at 60 seconds. **(A)** Estimated y- and z- head axis angular velocity components **(B)** Estimated and Actual pitch angle.

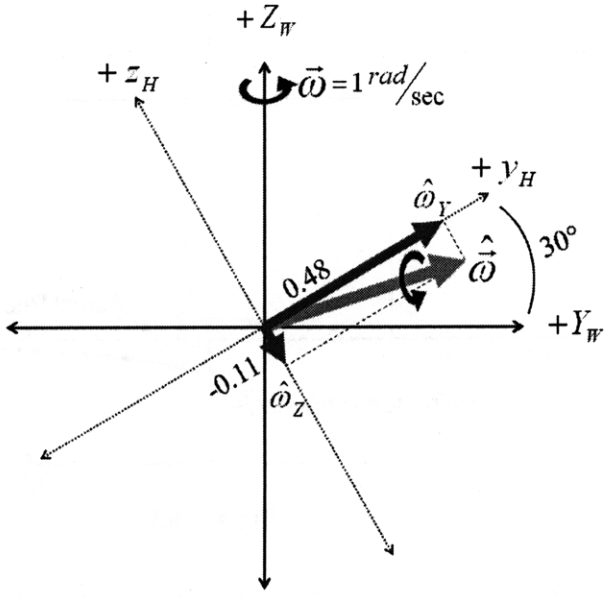


Figure 12. Vector analysis for simulated Coriolis Illusion pitch down sensation.

The Guedry and Benson analysis predicted the axis of tilt sensation, and the time course of angular velocity based on semicircular canal cues, but did not consider interaction with gravireceptor cues, or predict the magnitude and duration of illusory pitch. It is clear from subject's descriptions that the perceived pitch angle is not the integral of perceived rotational velocity. Instead, the resulting sensation is one of continuous tumbling, but limited tilt. The Observer model is able to successfully mimic this paradoxical sensation.

4.6 Pseudo Coriolis Illusion

Pitching or rolling head movements made during pure wide field of view optokinetic rotational stimulation about a vertical axis have also been found to be disorienting and nauseogenic (Dichgans et al 1973, Bles 1998, Johnson et al 1999, Brandt et al 1971). This has been traditionally referred to as the pseudo Coriolis illusion (Dichgans et al 1973, Bles 1998, Johnson et al 1999). In this simulation we use the Observer model to predict the pseudo Coriolis illusion and compare the direction and dynamics of the response with that of the Coriolis response, described in the previous *Section (4.5)*. Note that the stimuli in these two cases were deliberately chosen to correspond: 30 degree head movement towards the right shoulder in both cases, counter-clockwise physical rotation in the case of Coriolis, and clockwise visual scene rotation in the case of pseudo-Coriolis.

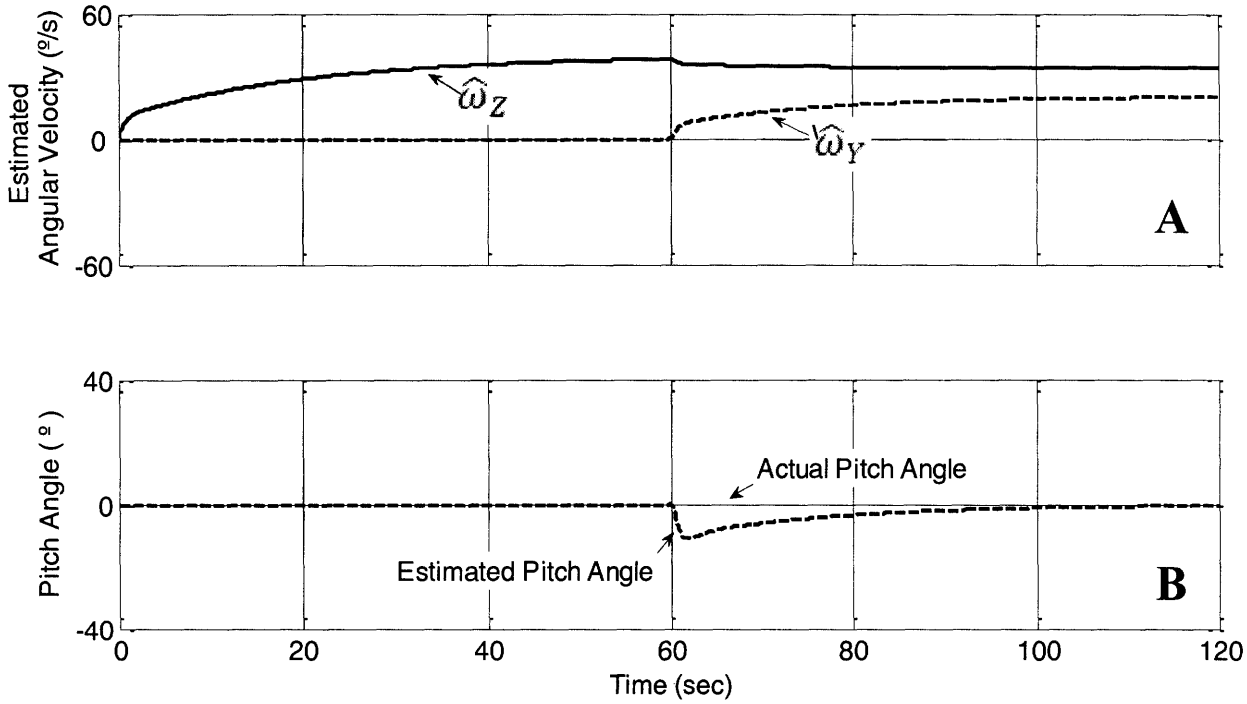


Figure 13. Simulation of pseudo-Coriolis. The simulated subject is placed in an optokinetic drum and remains physically stationary. The drum rotates clockwise at -1 rad/sec (-57.3°/s) about an Earth vertical axis for the duration of the simulation. At 60 seconds the subject makes a rolling head tilt of +30° (60°/s for 0.5 seconds) towards the right shoulder. The subject maintains this tilted head orientation for the remainder of the simulation. **(A)** Estimated y- and z- head axis angular velocity components **(B)** Estimated and Actual pitch angle.

The model predicts a transient pitch–up sensation (Figure 13B). Note that the direction of the illusory pitch sensation is opposite to the direction of the classic Coriolis illusion (Figure 11B). The estimated pitch angle reaches a peak value of -10.94° - much smaller than the corresponding Coriolis induced pitch sensation - and decays at a rate slightly slower than that of the Coriolis response (50 seconds vs. 40 seconds).

Note that the Observer model predicts that unlike the classic Coriolis Effect, pseudo Coriolis is not induced from a cross coupling of the semicircular canals. Rather, the Observer model predicts that the pseudo Coriolis illusion is the result of a fundamentally different neurological mechanism which could be described as "visual velocity storage".

In the Coriolis simulation, the estimated (perceived) yaw velocity sensation ($\hat{\omega}_Z$) has decayed to essentially zero prior to the rolling head tilt. When the head then is rolled away from the gravitational vertical, the horizontal semicircular canal experiences a sudden deceleration, and the vertical canals experience a corresponding acceleration. As noted by Guedry and Benson (1978), the resultant angular velocity components ($\hat{\omega}_Y, \hat{\omega}_Z$) combine to generate the pitch-down sensation predicted by the vector plots in Figure 12.

During pseudo Coriolis stimulation, the clockwise rotation of the visual scene maintains an illusory sensation of rotational motion – in the present case in a counterclockwise direction. As the head tilts toward the right shoulder, the unstimulated horizontal semicircular canal does not register any inertial deceleration and thus does not alter the magnitude or direction of the estimated yaw velocity ($\hat{\omega}_Z$). A perfect visual orientation estimator might respond to the tilted visual scene immediately and decompose $\hat{\omega}_Y$ and $\hat{\omega}_Z$ correctly with respect to the world coordinate frame. Under such conditions no illusory pitching sensation would be perceived. The Observer model posits, however, that the visual system does not respond instantaneously to the change in relative orientation of the optokinetic stimulus. Instead, as the head is tilted, the visual system stores the angular velocity estimate ("visual velocity storage") with respect to the head fixed coordinate frame and rotates this vector into the new 30° orientation. This produces a pitch up/tilt backward illusion along with a continued sensation of rotational motion. As the visual system starts to respond to the new multi-axis optokinetic input, the estimated angular velocity vector ($\hat{\vec{\omega}}$) begins to realign with the gravitational vertical. The time course and dynamics of the realignment process are governed by the fast and slow rise components of the circular vection onset response. Figure 14 shows the estimated angular velocity vector plots at three separate times during the course of the pseudo Coriolis illusion, as predicted by the Observer model.

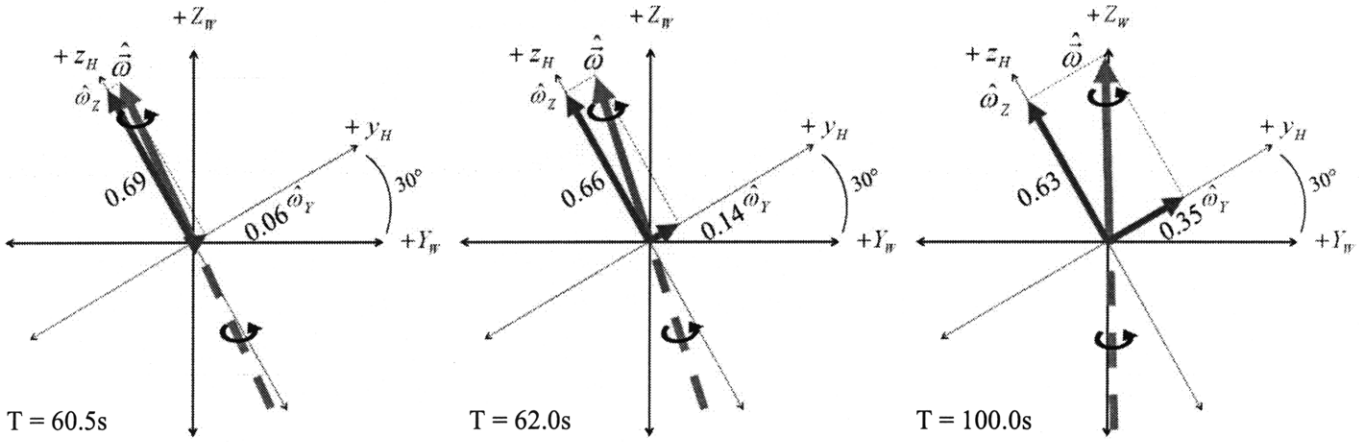


Figure 14. Head frame vector analysis for Observer model pseudo-Coriolis Illusion simulation. At 60.5 seconds, immediately following the head tilt, the visual velocity storage system has rotated the entire estimated angular velocity vector ($\hat{\vec{\omega}}$) into the head coordinate frame. At 62.0 seconds the fast component of circular vection is complete and the estimated angular velocity vector ($\hat{\vec{\omega}}$) is starting to align with the gravitational vertical. At 100 seconds the vector is almost parallel to the gravitational vertical and the subject experiences no more pitching sensation.

Several studies have investigated the nauseogenic properties of pseudo Coriolis (Dichgans et al 1973, Bles 1998, Johnson et al 1999). Dichgans et al (1973) and Bles (1998) report that vestibular Coriolis and pseudo Coriolis are qualitatively similar and proceed in the same direction:

"A model that would explain the pseudo Coriolis effects entirely, including the surprising conformity of direction of the illusory tilt in CE (Coriolis effect) and PCE (pseudo-Corilois effect) cannot yet be proposed" Dichgans et al (1973).

Both of these studies utilized magnitude estimation to compare the motion sickness provocative characteristics of Coriolis and pseudo Coriolis illusions but did not document the direction and time course of perceptions to back up their assertion that the direction of illusory tilt was similar. The Observer model simulations predict that the origins of the two illusions are fundamentally different, and the sign of the illusory tumbling and tilt should be in opposite directions.

A simple pilot study was conducted in the MIT Man Vehicle Lab (MVL) to determine the direction of the pseudo-Coriolis illusory response. Four subjects (ages 23 to 65) with no history of vestibular abnormality and with normal corrected vision were used. Subjects were seated upright in a stationary chair and positioned so that their head and upper torso were encased in a rotating polka-dotted (1 cm colored polka dots) optokinetic drum 50 cm in diameter. The drum was illuminated from above and rotated CCW in yaw at 36 degrees per second. Subjects were instructed to look straight ahead and indicate when the circular vection sensation had saturated to a maximum value. The direction of illusory yaw rotation was recorded. The subjects were then told to make an approximate 20 degree left ear down (LED) roll head tilt (-),

hold the orientation, and report the direction (pitch up / pitch down) of any resultant pitching sensation. Upon completion, the subject returned to an upright orientation and the experimental procedure was repeated for a right ear down (RED) roll tilt (+). 4 subjects, 3 naïve to the experiment, participated in the pilot study. (Note that the CCW direction of drum motion used in the pilot is opposite to the case illustrated above. The Observer model predicts that CCW drum rotation should result in CW circular vection sensation, and that when the head is tilted RED, a pitch down sensation should result. When a LED head movement is made, the sensation should be pitch up.)

TABLE 3. Directional Responses to pseudo-Coriolis

Subject #	Direction of CV Sensation	RED Tilt (+) Pitching Sensation	LED Tilt (-) Pitching Sensation
1	CW	Pitch Down	Pitch Up
2	CW	Pitch Down	Pitch Up
3	CW	Pitch Down	Pitch Up
4	CW	Pitch Down	Pitch Up

All four subjects reported a pitch up sensation for the LED roll head tilt and a pitch down sensation for the RED roll head tilt, Table 3. The direction of the responses match Observer model predictions for pseudo Coriolis, and are opposite to the Guedry and Benson Coriolis vector analysis and the Observer model's estimated Coriolis pitch angle results.

These results indicate that the reported direction of illusory pitch for the pseudo-Coriolis effect is indeed opposite to the classic Coriolis response. Although this was a pilot study, and additional naïve, vestibularly normal subjects should be tested before the results can be considered definitive, it provides the first experimental data recorded on the direction of the pseudo-Coriolis illusion. The term "pseudo-Coriolis" was originally proposed by Brandt probably because of the "cross coupled" nature of the sensation. However given that the sensations produced are in the opposite direction to those of vestibular Coriolis, and the mechanism involves visual velocity storage in head fixed axes, one could argue the term "pseudo-Coriolis" is a misnomer. A more appropriate term, given the predicted mechanism responsible for the phenomenon, would be "visual velocity storage cross coupling".

4.7 Tilt-Gain and OTTR/Tilt-Translation Illusions

Astronauts exposed to prolonged durations of weightlessness often experience landing vertigos (e.g. tilt-gain and tilt-translation illusions) upon return to Earth (Reschke et al 1987, Richards et al 2001, Young et al 1984). Two common and particularly disorienting effects are the Tilt-Gain and Tilt-Translation illusions, illustrated schematically in Figure 15.

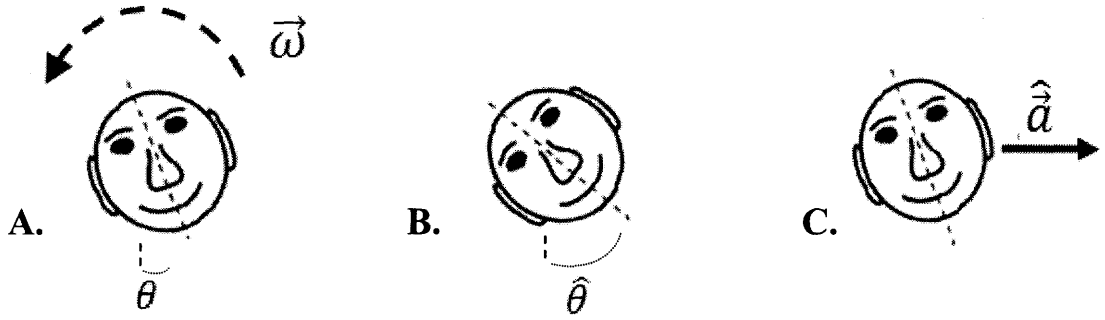


Figure 15. Tilt-Gain and Tilt-Translation Illusions. **(A)** Actual head tilt: Both illusions result from a small roll or pitch head tilt (θ). **(B)** Tilt – Gain Illusion perceived head tilt: Head tilt produces a tumbling sensation and a feeling that the head has rotated through an angle much greater angle than it actually has ($\hat{\theta}$). **(C)** Tilt – Translation Illusion perceived head tilt: Head tilt produces an illusory sensation of acceleration ($\hat{\vec{a}}$) and finite translation in the opposite direction of head tilt.

Merfeld (2003) attributed the illusory sensations to a central reinterpretation of rotational information (Rotational otolith tilt-translation reinterpretation (ROTTR) hypothesis). He postulated that human neurological ability to utilize rotational cues for the disambiguation of tilt and translation deteriorates in 0-G, and thus alters our ability to estimate gravity upon return to Earth. These misestimations in gravity, or "down," result in illusory sensations of acceleration as predicted by the vector plots in Figure 16.

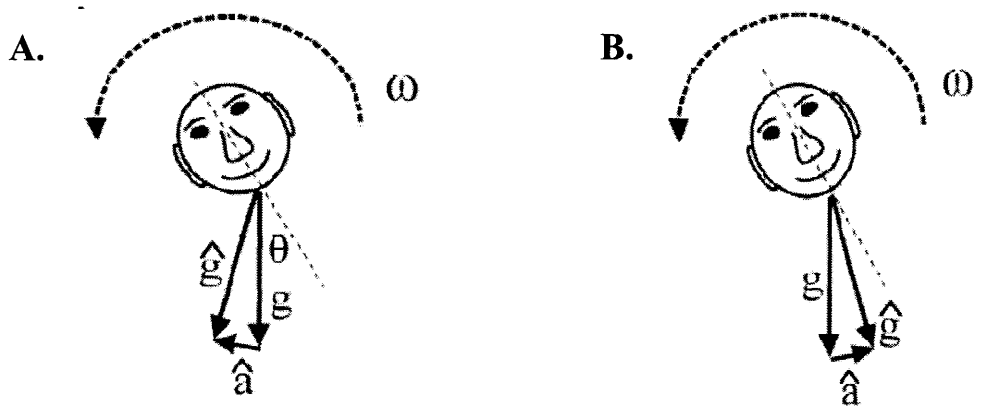


Figure 16. ROTTR hypothesis vector analysis. **(A)** Over estimation of head rotation based on angular cues causes an illusory sensation of linear acceleration in the same direction of head tilt. **(B)** Under estimation of head tilt causes a resultant linear acceleration and illusory sensation of translation in opposite direction. $\hat{a}$ - estimated acceleration, $\hat{g}$ - estimated gravity direction, g - actual gravity direction.

To simulate these illusions the residual weighting parameters of the internal CNS model must be modified to reflect the dynamics of a 0-g adapted astronaut. We assumed that exposure to a micro or zero-g environment alters the sensitivity and processing of rotational (ROTTR hypothesis, Merfeld 2003) information within the CNS. From a modeling perspective these assumptions translate to the modification of three individual free parameters ($K_{f\omega}$, K_f, $K_{\omega f}$). Given the lack experimental data on Tilt-Gain and Tilt-Translation[6], these parameters were tuned to qualitatively match the anecdotal accounts of the recorded post flight illusions following MIR Phase One (Richards et al 2001). Table 4 displays the standard Observer weighting values along with Tilt-Gain and Tilt-Translation 0-G adapted weighting schemes.

TABLE 4. *0-G Adapted Residual Weighting Parameters*

Parameter	Standard Values	Tilt-Gain Illusion	Tilt-Translation Illusion
$K_{f\omega}$	8	40	2
K_f	4	40	2
$K_{\omega f}$	1	10	0.1

Results (Figure 17) show that the modified model is able to mimic the characteristic of both the Tilt-Gain (Figure 17 A-C) and Tilt Translation (Figure 17 D-F) illusions. Over (Tilt-Gain) and under (Tilt-Translation) estimations of head tilt produce illusory sensations of linear acceleration as predicted by the ROTTR Hypothesis (Merfeld 1993). The Tilt-Gain response shows a peak over estimation of 33° (Figure 17A) which decays in 0.8 seconds and produces a small linear acceleration perception (Figure 17B) in the same direction of head tilt.

[6] Tilt-Gain and Tilt-Translation Illusion are typically only reported on the day of return from orbit and before vestibular investigators are able to extensively test subjects.

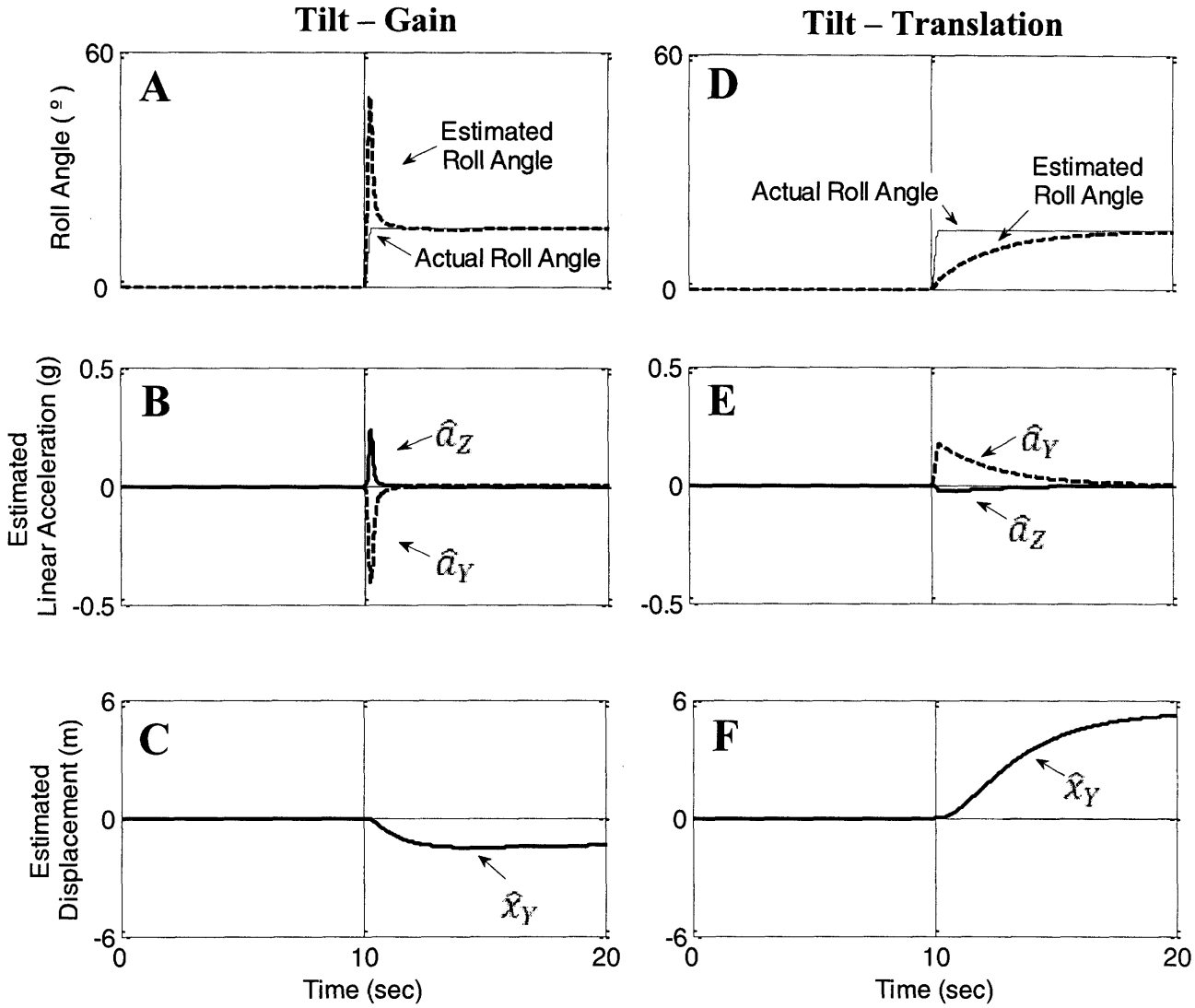


Figure 17. Model response to a +15 degree roll head tilt (executed in 0.2 seconds) for the Tilt – Gain residual weighting parameter scheme **(A, B, C)** and the Tilt – Translation residual weighting parameter scheme **(D, E, F).** For both simulations the head tilt was performed in lighted conditions and the simulated subject received visual linear and angular velocity cues as well as static visual position information. **(AD)** Estimated and actual roll angle. **(BE)** Estimated y- and z- head axis linear acceleration. **(CF)** Estimated x- world axis linear displacement ($\hat{x}_x$).

The Tilt Translation response shows an initial tilt under estimation of 13° (Figure 17D) which decays in 11.2 seconds and produces a large illusory linear acceleration (Figure 17E) in the opposite direction of head tilt. The resultant acceleration vector induces a sensation of lateral translation with a peak amplitude of 5.2 meters (Figure 17F). Verbal descriptions provided by returning astronauts and published by Richards et al (2001) provide us with an approximate description of the illusion.

> *"A classic tilt-translation illusion was my dominant vestibular effect upon return. When I tilted my head to the right, I felt I was translating to the left through a distance so large I thought I was in the next room."*

This description roughly correlates to the 5.2m predicted model response. Note that the Observer model also predicts that both illusions are transient and subside in relatively short durations of time. The ROTTR hypothesis was not able to explain why the illusions are temporary. According to ROTTR, the illusory sensations of lateral acceleration should proceed indefinitely while the subjects head is tilted. This prediction does not agree with typical (e.g. Richards et al (2001) description of the Tilt-Translation illusion.

> *"It was a persisting feeling while my head was tilted, but subdued by vision and knowledge that 'this can't be happening."*

Hence the Observer model accounts for why the illusions begin to subside due to a realignment of the gravity vector driven by vestibular dynamics. It also predicts that the time course of the illusions should depend on visual cues and predicts a saturation of the physical displacement due to visual position and velocity interactions.

Note that the model predicts that Tilt Gain and OTTR responses will be different if the subject has his eye closed and is tilted passively – an experiment that has not yet been done. For example, the tilt-gain model predicts that the subjects will dramatically over estimate head tilt, and then underestimate it. The sensation of motion is one of accelerating out and then rubber-banding back. Opening the eyes quenches the underestimation phase and makes the subject only feel acceleration in 1 direction. Flight surgeons who document Tilt-Gain and OTTR illusions in astronauts post flight should be certain to record whether the astronaut crewmember's eyes were open or closed.

CHAPTER 5. CONCLUSIONS AND RECOMMENDATIONS

We developed a multisensory Observer model for human spatial orientation perception. The model was successfully implemented and used to predict the responses to a number of visual and visual-vestibular motion paradigms. The vestibular only aspects of our interaction model largely derive from Merfeld's 1993 multidimensional spatial orientation model (Merfeld, et al 1993) and subsequent structural and parameter value refinements by Haslwanter et al (2000), Merfeld and Zupan (2002), and Vingerhoets et al (2006, 2007). Further extensions were required to obtain the necessary state estimates for visual sensory interaction and to capture the qualitative characteristics of large physical scale integrated self motion. First, we hypothesized that central integration of linear and angular motion – including azimuth - occurs in a perceptual world coordinate frame defined by the estimated direction of gravity. Perceived azimuth was not previously accounted for in other modeling efforts and was calculated via quaternion mathematics from the internal gravitational state. Second, model acceleration estimates were transformed to this "limbic" coordinate frame and integrated twice to obtain estimates of position and linear velocity. Leaky dynamics were assumed for the acceleration to velocity integration with individual time constants about each limbic coordinate axis. Third, the time constants were set to reflect qualitatively accurate path integration in the perceived horizontal "navigation plane" and mimic the large perceptual errors for vertical motion observed experimentally. Finally, two additional feedback gains were added to obtain finer model control and refine the loop gain of the angular velocity feedback loop. With these extensions in place the modified vestibular model was validated against a large set of classic vestibular motion paradigms.

1. Linear and angular acceleration steps
2. Postrotatory tilt
3. Constant velocity Earth vertical yaw rotation
4. Somatogravic illusion
    a. Forward linear acceleration on a sled
    b. Fixed and variable radius centrifugation
5. Static and dynamic roll tilt
6. Off-vertical-axis rotation (OVAR)
7. Large amplitude horizontal and vertical sinusoidal displacements

As expected, the model predicts accurate perceptual responses for the experimental simulations considered and validated by Merfeld (1-6). In contrast to the OVAR motion characteristics described by Denise et al (1988) and Wood et al (2002, 2007) (translatory motion about a conical or cylindrical path with a constant azimuth angle), the model predicts a sustained progression of perceived azimuthal direction, or heading. See recommendations below for a validation experiment for this finding. The model was also able to properly account for the large phase and magnitude estimation errors witnessed experimentally (Walsh 1964, Malcolm & Jones 1973, Jones et al 1978) for large amplitude, low frequency vertical motion. With the validated vestibular model in place a visual-vestibular interaction model was then proposed.

Visual – Vestibular Interaction Model

Visual information was added to the extended vestibular model in a manner consistent with the Merfeld (1993) topology. Four visual sensors were hypothesized to process both static (position and gravity/"down") and dynamic (linear and angular velocity) visual input. Sensory output was compared with an expected value from an internal CNS model and the resultant sensory conflict signal was weighted and added to the rate of change of the respective state. This new configuration necessitated four new residual weighting coefficients (K_{gv}, $K_{\omega v}$, K_{xv}, $K_{\dot{x}v}$) which were left as free parameters of the model. These parameters were tuned to match the modeling results of Borah, Young and Curry (1978) and the subsequent data sets they considered for their KF validation. Results were additionally compared to the Pommellet EKF (1990) simulations.

8. Constant velocity Earth vertical yaw rotation in the light
9. Circular vection
10. Somatogravic illusion due to forward linear acceleration in a lighted cabin
11. Forward linear vection

The model predicts a sustained sensation of rotation in the light and a gradual onset of circular vection sensation in response to pure optokinetic drum rotation. The dynamic response of circular vection indicates two individual components responsible for the onset and saturation of the perceived rotation. These components correlate to the fast and slow rise circular vection components witnessed experimentally during OKN and OKAN (Cohen et al 1981, Jell et al 1984, Lafortune et al 1986) and were not predicted by the Borah KF or Pommellet EKF models. The model predicts that the "pitch up" illusion during forward linear acceleration in the light is suppressed due to visual gravity/"down" cues and not visual linear vection information as modeled by Borah and Pommellet. This prediction is consistent with the findings of Tokumaru et al (1998). Vection onset delays could not be properly modeled with the Borah or Observer model; however our interaction model was able to account for the quick vection rise time reported by Chu (1976) which the KF model failed to reproduce. Finally the predictive capability of the Observer model was tested with several more complex visual-vestibular sensory paradigms.

12. Coriolis
13. Pseudo Coriolis
14. Astronaut post-flight illusions
    a. Tilt-Gain
    b. Tilt-Translation

The model correctly predicts both the direction of the Coriolis Effect, and also the magnitude of the resulting tilt illusion. It is also able to account for the reported paradoxical

sensation of continued tumbling with limited tilt. In addition to Coriolis, a pseudo Corilois illusion was simulated. Pseudo Coriolis results from a pitching or rolling head movements made during pure wide field of view optokinetic rotational stimulation about a vertical axis, and has been described to be both "qualitatively similar" and to "proceed in the same direction" as the Coriolis illusion (Dichgans et al 1973, Bles 1998). The model predicts that pseudo Coriolis illusion is due to a fundamentally different mechanism ("visual velocity storage") than the Coriolis illusion, and the resulting illusory pitching sensations occur in an opposite direction. The perceived illusory pitch angle duration is similar although reduced in magnitude to the Coriolis response. These predictions were tested and verified with a pseudo Coriolis illusion pilot experiment. All four subjects reported illusory pitching sensation of pseudo Coriolis in the opposite direction of a Coriolis response. These findings offer the first experimental data on the direction of the pseudo Corilois illusion and suggest that a more appropriate term for the response, given the predicted mechanism responsible, would be "visual velocity storage cross coupling." Finally, the model was used to reproduce astronaut post flight illusions (Tilt-Gain and Tilt-Translation) following prolonged exposure to weightlessness and return to Earth. In order to simulate a 0-G adapted astronaut the internal residual weighting parameters were modified in a manner consistent with the ROTTR hypothesis (Merfeld 2003) (e.g. increase/decease of rotational cue sensitivity for the disambiguation of tilt and translation). The model was able to account for the dynamics of both illusions as described by the ROTTR theory (e.g. Tilt-Gain – Over estimation of head tilt and translational sensation in same direction as tilt. Tilt-Translation – Underestimation of head tilt and translational sensation in the opposite direction). The Model was additionally able to explain why the illusions are transient, as reported anecdotally, instead of sustained as predicted by the ROTTR theory.

All simulations were performed in the OBSERVER Spatial Orientation Analysis Tool. We developed this tool to aid in the processing, simulation and visualization of human perception in response to 3D, complex, multisensory motion stimuli. The program includes Excel spreadsheet input capability and a GUI to make the model accessible to less expert Matlab users. Orientation and motion predictions can be plotted in 2D or visualized in 3D using virtual avatars. OBSERVER was designed to be more easily used by sensorimotor investigators, human factors engineers and by disorientation incident/accident investigators.

In summary, the Observer visual-vestibular interaction model, and the OBSERVER Spatial Orientation Analysis Tool, were successfully implemented and used to predict the responses to a number of visual and visual-vestibular motion paradigms. Model predictions generally exceeded the Borah KF and Pommellet EKF results both in terms of accuracy, numerical stability and general scope of application. By extending the traditional Observer structure to include estimates of azimuth, position, and velocity, and incorporating sensory information from four independent visual sources, we have dramatically broadened the range and depth Observer based models can provide in orientation analysis. Additionally, the model provides new explanations and predictions for perceptual phenomenon that have either been

previously untested (Tilt-Gain and Tilt-Translation Illusion) or not fully investigated and understood (pseudo Coriolis).

Although the model was successful in meeting the original five modeling goals, several limitations exist and require additional attention and refinement. For example, sensory noise could be added and used to model perceptual thresholds of the canals and otoliths. These thresholds are important in the dynamics of many classic aeronautical flight illusions (e.g. the leans and graveyard spiral). Also, additional visual system nonlinearities and dynamics could be incorporated to account for saturation limits, vection delays and other characteristics of focal and periphery vision. Finally, to achieve our secondary goal of incorporating an Observer model in the Alion Corp. SDAT program, the Simulink model should be recoded using Matlab toolbox functions, and compiled into an executable file. (This is necessary since the Simulink code currently used in the model currently cannot be compiled).

Several pertinent experiments are required to further validate Observer model predictions and structure. First, the pilot experiment on pseudo-Coriolis should be repeated with a larger number of subjects. The experiment should record and directly compare and the direction, time course, and magnitude of resulting illusory tilting sensations for both pseudo-Coriolis and Coriolis stimulation.

In order to validate the limbic coordinate frame velocity and displacement integrator aspects of the model, a large amplitude vertical vs. horizontal motion experiment is required. The experiment should test vertical motion with the human body Z axis aligned with and perpendicular to the gravity axis, measuring perceived displacement amplitude and phase. An identical experiment could be preformed for horizontal motion with the body z-axis aligned with and perpendicular to the direction of displacement. To validate the azimuth aspects of the limbic frame a simple OVAR experiment could be designed which specifically targets the subject's perceived direction of azimuth.

While access to post flight astronauts is highly limited, the broadening possibility of commercial space flight and space tourism opens potential opportunities to test aspects of the Tilt-Gain and Tilt-Translation Illusions. A simple experiment could involve a seated subject active and passively tilted in both directions. The direction, magnitude, duration, and perceived velocity of any resulting illusory sensation could be recorded for both tilting conditions with eyes open and eyes closed. The experiments could also be performed at several different times post flight to indicate the time course of decay of the illusions.

REFERENCES

Aoki, H., Ohno, R., and Yamaguchi, T., 2003, A study of spatial orientation in a virtual weightless environment. Part 2 Causes of spatial cognition errors, *Journal of Architecture, Planning, and Environmental Engineering* **563**:85-92.

Aoki, H., Ohno, R., Yamaguchi, T., 2005, The effect of the configuration and the interior design of a virtual weightless space station on human spatial orientation, *Acta Astronautica* **56**:1005-1016.

Berthoz, A., Pavard, B., and Young, L.R., 1975, Perception of linear horizontal self-motion induced by peripheral vision (linearvection): Basic characteristics and visual-vestibular interactions, *Experimental Brain Research* **23**:471-489.

Best, P. J., White, A. M., and Minai, A, 2001, Spatial processing in the brain: the activity of hippocampal place cells, *Annual Review of Neuroscience* **24**:459-86.

Bilien, V., 1993, Modeling human spatial orientation perception in a centrifuge using estimation theory, S.M. thesis, Man-Vehicle Laboratory, Cambridge, Massachusetts: Massachusetts Institute of Technology.

Bles, W., 1998, Coriolis effect and motion sickness modeling, *Brain Research Bulletin* **47**:543-549.

Borah, J., Young, L.R., Curry, R.E., 1978, Sensory mechanism modeling, Air Force Human Resources Laboratory, Air Force Systems Command, AFHRL-TR-78-83.

Borah, J., Young, L.R., Curry, R.E., 1988, Optimal estimator model for human spatial orientation, *Annals of the New York Academy of Sciences* **545**:1-73.

Brandt, Th., Wist, E., Dichgans, J., 1971, Optisch induzierte Pseudocoriolis-Effekten und Circularvektion. *Arch. Psych. Nervenkrank.* **214**:365–389.

Calton, J. L., and Taube, J.S., 2005, Degradation of head direction cell activity during inverted locomotion, *J Neurosci* **25**:2420-2428.

Chu, W., Dynamic response of human linearvection, 1976, S.M. thesis, Department of Aeronautics and Astronautics, Cambridge, Massachusetts: Massachusetts Institute of Technology.

Cohen, B., Henn, V., Raphan, T., Dennett, D., 1981, Velocity storage, nystagmus and visual-vestibular interactions in humans, *Annals of the New York Academy of Sciences* **374**:421-433.

Cohen, M.M., Crosby, R.H., and Blackburn, L.H., 1973, Disorienting effects of aircraft catapult launchings, *Aerospace Medicine* **44**:37-39.

Dai, M.J., Curthoys, I.S., Halmagyi, G.M., 1989, A model of otolith stimulation, *Biological Cybernetics* **60(10)**:185-194.

Denise, P., Darlot, C., Droulez, J., Cohen, B., Berthoz, A., 1988, Motion perceptions induced by off-vertical axis rotation (OVAR) at small angles of tilt, *Experimental Brain Research* **73**:106–114.

Dichgans, J., Brandt, Th., 1973, Optokinetic motion sickness and pseudo-Coriolis effects induced by moving visual stimuli, *Acta Otolaryngica (Stockh).* **76**:339-348.

Fang, A.C., and Zimmerman, B.G., 1969, Digital simulation of rotational kinematics, Washington D.C: Goddard Space Flight Center; NASA TN D-5302:1-27.

Graybiel, A., Brown, R.,1951, The delay in visual reorientation following exposure to a change in direction of resultant force on a human centrifuge, *Journal of General Psychology* **45**:143-150.

Graybiel, A., 1966, Orientation in aerospace flights. Joint Report, Naval Aerospace Medical Institute, NASA. Special Report 66-6, NASA order R-93. Pensacola, FL.

Groen, E.L., Smaili, M.H., and Hosman, R.J.A.W., 2007, Perception model analysis of flight simulator motion for a decrab maneuver, *Journal of Aircraft* **44(2)**:427-435.

Guedry, F.E., Jr., and Benson, A.J., 1978, Coriolis cross-coupling effects: Disorienting and nauseogenic or not?, *Aviation, Space and Enviromental. Medicine* **49(1)**:29-35.

Guedry, F.E., Jr., and Harris, C.S., 1963, Labyrinthine function related to experiments on the parallel swing, NASA Joint Report No. 86 Bureau of Medicine and Surgery MR005.13-6001.

Hafting, T., Fyhn, M., Molden, S., Moser, M. B., and Moser E. I., 2005, Microstructure of a spatial map in the entrohinal cortex, *Nature* **436**:801-806.

Haslwanter, T., Jaeger, R., Mayr, S., Fetter, M., 2000, Three-dimensional eye-movement responses to off-vertical axis rotations in humans, *Experimental Brain Research* **134**:96-106.

Israël, I., Grasso, R., Georges-François, P., Tsuzuku, T. and Berthoz, A., 1989, Spatial Memory and Path Integration Studied by Self-Driven Passive Linear Displacement. I. Basic Properties, *Journal of Neurophysiology* **77:**31803192.

Jell, R., Ireland, D., Lafortune, S., 1984, Human optokinetic afternystagmus. Slow-phase characteristics and analysis of the decay of slow phase velocity, *Acta Otolaryngolgica (Stockh.)* **98**:462-471.

Johnson, W.H., Sunahara, F.A., and Landolt, J.P., 1999, Importance of the vestibular system in visually induced nausea and self-vection, *Journal of Vestibular Research* **9**:83–87.

Jones, G.M., Young, L.R., 1978, Subjective detection of vertical acceleration: A velocity dependent response?, *Acta Otolaryngologica (Stockh.)* **85**:45-53.

Knierim, J. J., McNaughton, B. L., and Poe, G. R., 2000, Three –dimensional spatial selectivity of hippocampal neurons during space flight, *Nature Neuroscience* **3**:209-210.

Knierim, J. J., Poe, G. R., and McNaughton, B. L., 2003, Ensemble neural coding of place in zero-g, in: *The Neurolab Spacelab Mission: Neuroscience Research in Space: Results from the STS-90 Neurolab Spacelab Mission*. J. C. Buckey, Jr., and J. L. Homick, eds., NASA SP-2003-535, pp.63-68.

Lafortune, S., Ireland, D., Jell. R., DuVal, L., 1986, Human optokinetic afternystagmus. Stimulus velocity dependence on the two-component decay model and involvement in pursuit, *Acta Otolaryngologica (Stockh.)* **1010**:183-192.

Loomis, J.M., Klatzky, R.L., Golledge, R.G., Cicinelli, J.G., Pellegrino, J.W., and Fry, P.A., 1993, Non-visual Navigation by Blind and Sighted: Assessment of Path Integration Ability, *Journal of Experimental Psychology* **122**:73-91.

Luenburger, D.G., 1971, An introduction to observers, IEEE *Transactions on Automatic Control* **16**:596-602.

Malcolm, R., and Melvill Jones, G., 1973, Erroneous perception of vertical motion by humans seated in the upright position, *Acta Otolaryngologica (Stockh.)* **77**:274-283.

McGrath, B.J., 2005, US Naval Aerospace Medical Research Laboratory, Pensacola, Personal communication with CM Oman.

Merfeld, D.M., Young, L.R., Oman, C.M., Shelhamer, M.J., 1993, A multidimensional model of the effects of gravity on the spatial orientation of the monkey, *Journal of Neurophysiology* **3**:141-161.

Merfeld, D.M., Zupan, L.H., 2002, Neural processing of gravitoinertial cues in humans. III Modeling tilt and translation responses, *Journal of Neurophysiology* **87**:819-833.

Merfeld, D.M., 2003, Rotation otolith tilt-translation reinterpritation (ROTTR) hypothesis: A new hypothesis to explain neurovestibular spaceflight adaptation, *Journal of Vestibular Research* **13**:309-320.

Mittelstaedt, M.L., Glasauer, S., 1991, Idiothetic navigation in gerbils and humans, *Zool J Physiol* **95**:427–435.

Mittelstaedt, M.L., Mittelstaedt, H., 2001, Idiothetic navigation in humans: estimation of path length, *Experimental Brain Research* **139**:318–332.

Oman, C.M., 1982, A heuristic mathematical model for dynamics of sensory conflict and motion sickness, *Acta Otolaryngologica (Stockh.)* **392***(Suppl)*:1-44.

Oman, C.M., 1991, Sensory conflict in motion sickness: an Observer Theory approach, *Pictorial communication in real and virtual environments* S. Ellis. London, Taylor and Francis: 362-367.

Oman, C.M., 2007, Spatial Processing in Navigation, Imagery and Perception, *Spatial Orientation and Navigation in Microgravity* Springer US: 209-247.

Parker, D. E., Reschke, M. F. , Arrott, A. P., Homickand, J. L., Lichtenberg, B. K., 1985, Otolith tilt-translation reinterpretation following prolonged weightlessness: implications for preflight training, *Aviation, Space, and Environmental Medicine* **56**:601-606.

Pommellet, P.E., 1990, Suboptimal estimator for the spatial orientation of a pilot, S.M. thesis, Man-Vehicle Laboratory, Cambridge, Massachusetts: Massachusetts Institute of Technology.

Raphan, Th., Matsuo, V., Cohen, B., 1977, A velocity storage mechanism responsible for optokinetic nystagmus (OKN), optokinetic after-nystagmus (OKAN) and vestibular nystagmus. In: Baker, R., Berthoz, A., eds. Control of gaze by brain stem neurons. Amsterdam: Elsevier/North-Holland Biomedical Press, 37-47.

Raphan, Th., Matsuo, V., and Cohen, B., 1979, Velocity storage in the vestibulo-ocular reflex arc (VOR), *Experimental Brain Research* **35**:229-248.

Raphan, T., Cohen, C., 2002, The vestibulo-ocular reflex in three dimensions, *Experimental Brain Research* **145**:1-27.

Reschke, M.F., and D.E., Parker, 1987, Effects of prolonged weightlessness on self-motion perception and eye movements evoked by roll and pitch, *Aviation, Space, and Environmental Medicine* **58(9)**:A153-A157.

Richards, J.T., Clark, J.B., Oman C.M., and Marshburn, T.H., 2001, Neurovestibular effects of long-duration spaceflight: A summary of Mir phase 1 experiences: NSBRI Report 2001 1-33.

Robinson, D.A., 1977, Vestibular and optokinetic symbiosis; an example of explaining by modeling. In: Baker, R., Berthoz, A., eds. Control of gaze by brain stem neurons. Amsterdam: Elsevier/North-Holland Biomedical Press, 49-58.

Seidman, S.H., 2008, Translational motion perception and vestiboocular responses in the absence of non-inertial cues, *Experimental Brain Research* **184**:13-29.

Small, R.L., Keller, J.W., Wickens, C.D., Socash, C.M., Ronan, A.M., Fisher, A.M., 2006, Multisensory integration for pilot spatial orientation, Micro Analysis and Design, Boulder Colorado Report A253074.

Tokumaru, O., Kaida, K., Ashida, H., Mizumoto, C., Tatsuno, J., 1998, Visual influence on the magnitude of somatogravic illusion evoked on advanced spatial disorientation demonstrator, *Aviation, Space, and Environmental Medicine* **69**:111-116.

Vidal, M., Lipshits, M., McIntyre, J., and Berthoz, A., 2003, Gravity and spatial orientation in virtual 3D-mazes, *Journal of Vestibular Research* **13**:273-286.

Vidal, M., Amorim, M.A., and Berthoz, A., 2004, Navigating in a virtual three dimensional maze: How do egocentric and allocentric reference frames interact? *Cognitive Brain Research* **19**:244–258.

Vingerhoets, R.A.A., Medendorp, W.P., Van Ginsbergen, J.A.M., 2006, Time course and magnitude of illusory translation perception during off-vertical axis rotation, *Journal of Neurophysiology* **95**:1571-1587.

Vingerhoets, R.A.A., Van Ginsbergen, J.A.M., Medendorp, W.P., 2007, Verticality perception during off vertical axis rotation, *Journal of Neurophysiology* **97**:3256-3268.

Waespe, W., and Henn, V., 1977, Neuronal activity in the vestibular nuclei of the alert monkey during vestibular and optokinetic stimulation, *Experimental Brain Research* **27**:523-538.

Walsh, E.G., 1964, The perception of rhythmically repeated linear motion in the vertical plane, *Experimental Physiology* **49:**58-65.

Wolfe, J.W., and Cramer, R.L., 1970, Illusions of pitch enduced by centripital acceleration, *Aerospace Medicine.* **41(10)**:1136-9.

Wood, S.J., 2002, Human otolith-ocular reflexes during off-vertical axis rotation: effect of frequency on tilt-translation ambiguity and motion sickness. *Neuroscience Letters* **323**:41–44.

Wood, S.J., Reschke, M.F., Sarmiento, L.A., Clement, G., 2007, Tilt and translation motion perception during off-vertical axis rotation, *Experimental Brain Researcb* **182**:365-377.

Young, L.R., Oman, C.M., Watt, D.G., Money, K.E., and Lichtenberg, B. K., 1984, Spatial orientation in weightlessness and readaptation to Earth's gravity, *Science* **225**: 205-208.

APPENDIX A. COORDINATE SYSTEMS AND SPATIAL ROTATIONS

A.1 Coordinate Systems

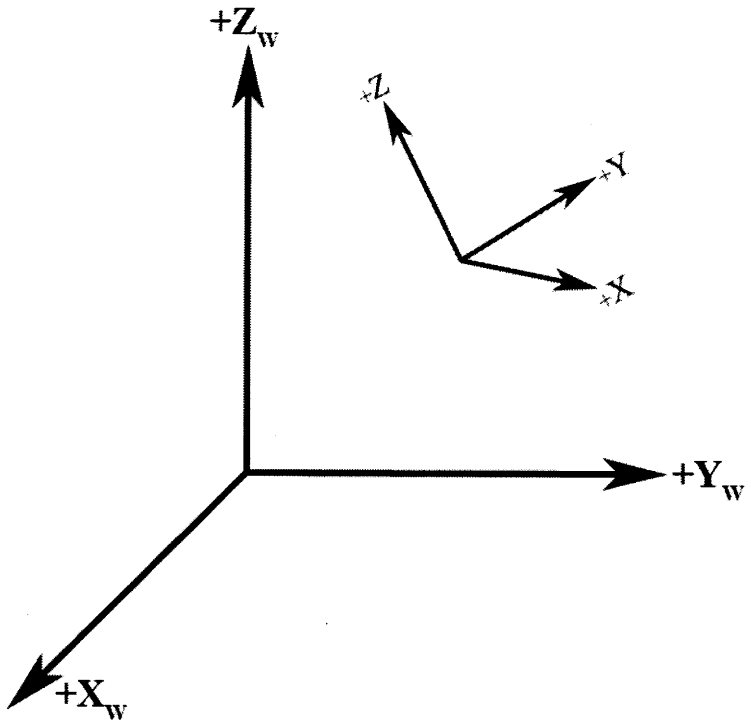


Figure A1. Head and World Coordinate Frames

We define a right handed coordinate system relative to the world (X_W, Y_W, Z_W) and the head (X, Y, Z), Fig A1. It is assumed that the semicircular canals and otoliths are situated at the center of the head and align with the naso-occiptal X, interaural Y, and dorsoventral Z, axes. For computational simplicity angular velocity and linear acceleration inputs to the OBSERVER vestibular model are processed in the egocentric head fixed frame. It is often necessary to transform quantities between reference frames. Gravity, for instance, is inherently defined in world coordinates yet needed for the GIF ("gravito-inertial force") calculation performed in the head axes. As we rotate and translate about in space a novel description of the relationship between these coordinate frames is therefore required.

A.2 Quaternion Representation

The quaternion provides us with a useful notation for representing spatial rotations. Quaternions eliminate gimbal lock, reduce numerical storage from 9 to 4 digits (9 being the typical representation of a rotation matrix), and increase computational stability. A quaternion representation for our model's coordinate frames and vectoral rotations is therefore preferred. We can define a unit quaternion in the following form:

$$\vec{q} = q_0 + q_1 i + q_2 j + q_3 k$$

$$\text{with} \quad i^2 = j^2 = k^2 = -1$$

In order to update the quaternion vector as we rotate in inertial space the initial quaternion $\vec{q}_0$[7] must be integrated with respect to the angular velocity input, $\vec{\omega}(t) = [\omega_X(t), \omega_Y(t), \omega_Z(t)]^T$. A stable algorithm to perform this integration was developed by Fang & Zimmerman (1969).

$$\dot{q}_0 = -\frac{1}{2}(q_2\omega_X + q_1\omega_Y + q_3\omega_Z) + k\lambda q_0 \tag{1}$$

$$\dot{q}_1 = -\frac{1}{2}(q_1\omega_X - q_0\omega_Y - q_2\omega_Z) + k\lambda q_1 \tag{2}$$

$$\dot{q}_2 = \frac{1}{2}(q_0\omega_X + q_3\omega_Y - q_1\omega_Z) + k\lambda q_2 \tag{3}$$

$$\dot{q}_3 = \frac{1}{2}(q_1\omega_X - q_2\omega_Y + q_0\omega_Z) + k\lambda q_3 \tag{4}$$

$$\lambda = 1 - (q_0^{\,2} + q_1^{\,2} + q_2^{\,2} + q_3^{\,2}) \tag{5}$$

Integrating Equations 1 – 5 yields a complete time history for the quaternion vector $\vec{q}$. This particular formulization uses an algebraic constraint to minimize the constraint error. For alternate integration schemes using normalization or derivative constraints one is referred to Fang & Zimmerman 1969. Constraint errors represent a non-orthonormaility in the transformation matrix and are thus extremely problematic for the decomposition of vectors. The proportionality constant "*k*" ensures stability such that $k > 0$ and the product $hk \leq 1$, where *h* is defined as the integration time step. A value of k = 0.8 – 0.9 worked best for our input file sample rates and Simulink ODE45 differential equation solver.

The integrated quaternion now provides us with all the necessary information to transform vectors between the head and world coordinate frames. At each time step a rotation of the gravity vector $\vec{g}_W = [g_{XW}, g_{YW}, g_{ZW}]^T$ is accomplished with the transformation matrix **T**;

[7] The initial quaternion is calculated based on the initial gravitational state. Assuming the subject is oriented upright, inline with the gravitational vertical, and in a 1G environment $\vec{g}_W = [0,0,-1]^T$ and $\vec{q}_0 = [1,0,0,0]^T$.

$$R=\begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_0q_1+q_2q_3) \\ 2(q_0q_2+q_1q_3) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix}$$

such that the current direction of gravity in head coordinates can be expressed as the premultiplication of $\vec{g}_W$ with **T.**

$$\vec{g}(t)=T(\vec{q})\cdot\vec{g}_W \tag{6}$$

Likewise, we can integrate Equations 1 – 5 with respect to the internal estimate of angular velocity, $\hat{\vec{\omega}}$, and obtain a similar representation for the estimated gravity state.

$$\hat{\vec{g}}(t)=T(\hat{\vec{q}})\cdot\hat{\vec{g}}_W \tag{7}$$

It should be noted that the estimated initial gravity vector, $\hat{\vec{g}}_W$, and the true initial gravity vector, $\vec{g}_W$, are assumed to be equivalent. A discrepancy between these vectors would result in a perceived sustained acceleration inconsistent with actual human perception. In any gravity environment, the CNS is therefore modeled to maintain an initially veridical estimate of the direction and magnitude of gravity.

For many aero and astronautical applications the input or desired output of vehicle trajectories are expressed in Euler Angles. Though dismissed for spatial rotations within the model, the ability to output and convert rotations to Euler angles is useful for both completeness and compatibility with a broader range of data sources. From the quaternion vector, the Euler Angles for a Z-Y-X (Yaw – Pitch – Roll) rotation sequence can be calculated using;

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix}=\begin{bmatrix}\arctan\dfrac{2(q_0q_1+q_2q_3)}{q_0^2-q_1^2-q_2^2+q_3^2}\\ \arcsin(2q_0q_2-q_1q_3)\\ \arctan\dfrac{2(q_0q_3+q_1q_2)}{q_0^2+q_1^2-q_2^2+q_3^2}\end{bmatrix}$$

where ϕ,θ,ψ, correspond to the roll, pitch, and yaw angle respectively. The Z-Y-X rotation sequence limits pitch angles between $\pm 90°$ and Roll & Yaw angles between $\pm 180°$.

A.3 Limbic Coordinate Frame Calculation and Quaternion Transformation

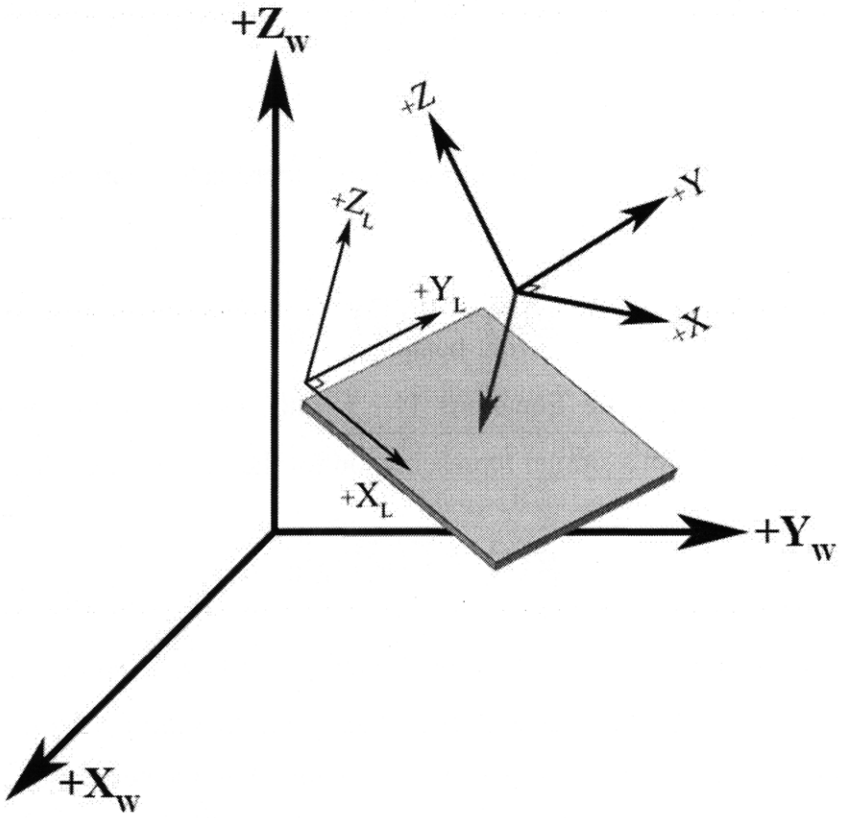


Figure A2. Limbic Coordinate Frame

The quaternion vector, $\hat{\vec{q}}$, from the estimated gravitational state completely defines the limbic coordinate frame. As shown in Figure A2. the limbic frame confines to the same right handed orientation and sign conventions of the other coordinate systems. The X_L-Y_L horizontal plane is perpendicular to the direction of estimated gravity and acts as our natural plane of 2D navigation. Just as **T** was used to transform vectors from world to head coordinates, its inverse can be used to perform the opposite duty. Driven by the estimated quaternion, $\hat{\vec{q}}$, $\mathbf{T}^{-1}$ can transform any vector from head to limbic coordinates. Rearranging Equation 7, and substituting the estimated acceleration vector in place of gravity we obtain;

$$T(\hat{\vec{q}}) \cdot \hat{\vec{a}}_L(t) = \hat{\vec{a}}(t) \tag{8}$$

Where $\hat{\vec{a}}_L(t)$ corresponds to the estimated acceleration vector expressed in the limbic frame. Pre-multiplying both sides of Equation 8 by the inverse transformation we can solve explicitly for $\hat{\vec{a}}_L(t)$.

$$T^{-1}(\hat{\vec{q}}) \cdot T(\hat{\vec{q}}) \cdot \hat{\vec{a}}_L(t) = T^{-1}(\hat{\vec{q}}) \cdot \hat{\vec{a}}(t)$$

$$\hat{\vec{a}}_L(t) = T^{-1}(\hat{\vec{q}}) \cdot \hat{\vec{a}}(t)$$

A.4 Visual position and velocity transformations

A direct conversion between the world and limbic frame is not possible with the current quaternion setup. To circumvent this potential problem inputs are converted from world to head coordinates and then subsequently from the head to the limbic system. This rotation if performed with two transformation matrices. The expressions for visual position and velocity are shown below.

$$T^{-1}(\hat{\vec{q}}) \cdot (T(\vec{q}) \cdot \vec{x}_V)$$

$$T^{-1}(\hat{\vec{q}}) \cdot (T(\vec{q}) \cdot \dot{\vec{x}}_V)$$

A.5 Visual gravity and angular velocity transformations

Visual gravity and angular velocity are transformed to head coordinates prior to processing. Rearranging equation 6 and substituting the visual inputs we obtain expressions for their coordinate frame transformations.

$$T(\vec{q}) \cdot \vec{\omega}_V$$

$$T(\vec{q}) \cdot \vec{g}_V$$

APPENDIX B. OBSERVER SPATIAL ORIENTATION ANALYSIS TOOL USER GUIDE

OBSERVER V1.2 User Guide

Michael Newman and Charles Oman
MIT Man Vehicle Laboratory
Last Updated May, 2009

OBSERVER v 1.2 is a Matlab/Simulink based tool developed to predict the time course of 3D human spatial orientation in response to complex motion stimuli. As compared to earlier research versions, OBSERVER is designed to be more easily used by sensorimotor investigators, human factors engineers and by disorientation incident/accident investigators. Though originally validated using 1-G human and animal data, the model has been extended to predict responses in 0-G, 1/6 (lunar)G and 3/8 (Mars) G, and the presence or absence of visual cues. It can mimic head movement contingent vertigos after spaceflight. OBSERVER was developed as part of a low-gravity spatial disorientation research project funded by the National Space Biomedical Research Institute, as a companion to SDAT, a disorientation prediction tool developed by Alion Science and Technology Corp. Input time series data are supplied via Excel spreadsheet, using a specific format. After each simulation, OBSERVER displays a family of 2-D plots of model inputs and outputs. A separate 3D visualization window dynamically displays the time course of observer model "down" and "azimuth" estimates. 3D VR simulations with virtual manikins also aid in visualization. Advanced users can easily archive specific model responses using conventional MATLAB commands.

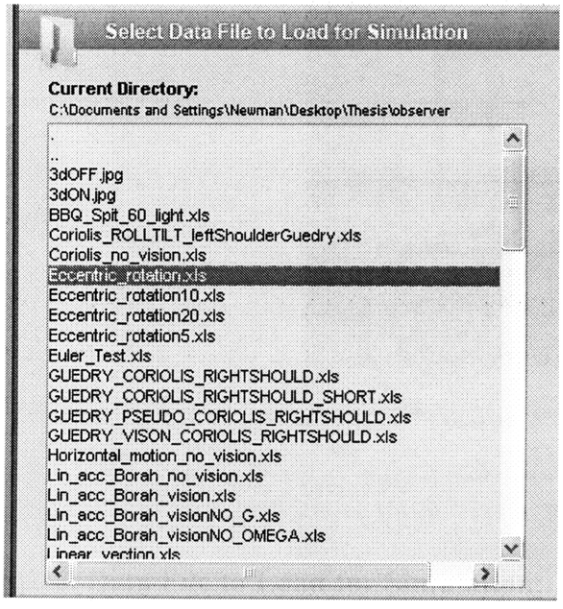
Getting started:

1. Copy all the OBSERVER model files provided into your My Documents\MATLAB folder or some other suitable location on the MATLAB search path.
2. Check the MATLAB desktop window Current Directory, and verify that it displays the files in the OBSERVER model folder.
3. (Optional) Examine the OBSERVER Simulink model by right clicking on the observerModel.mdl file in the MATLAB Current Directory window, and selecting OPEN. See Model Overview section below for a tour of the model.
4. Open the OBSERVER model control window by either:
    a. typing observer.m <enter> into the MATLAB command window after the >> prompt, or
    b. right clicking on observer.m in the MATLAB current directory window, and selecting "run".

Observer Interface:

All features of the Observer 1.2 spatial orientation modeling tool can be accessed from the main GUI window. Each feature is highlighted below.

Select Data File to Load for Simulation: Double click on one of OBSERVER example .xls data files (e.g. OVAR90.xls) in the OBSERVER "Current Directory" window. When the file loads, the control panel buttons will activate.

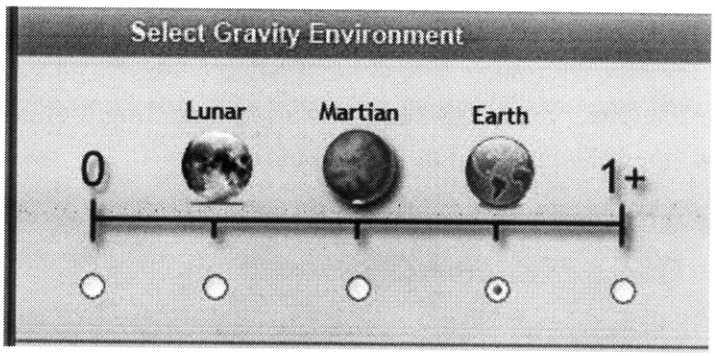


Notes:

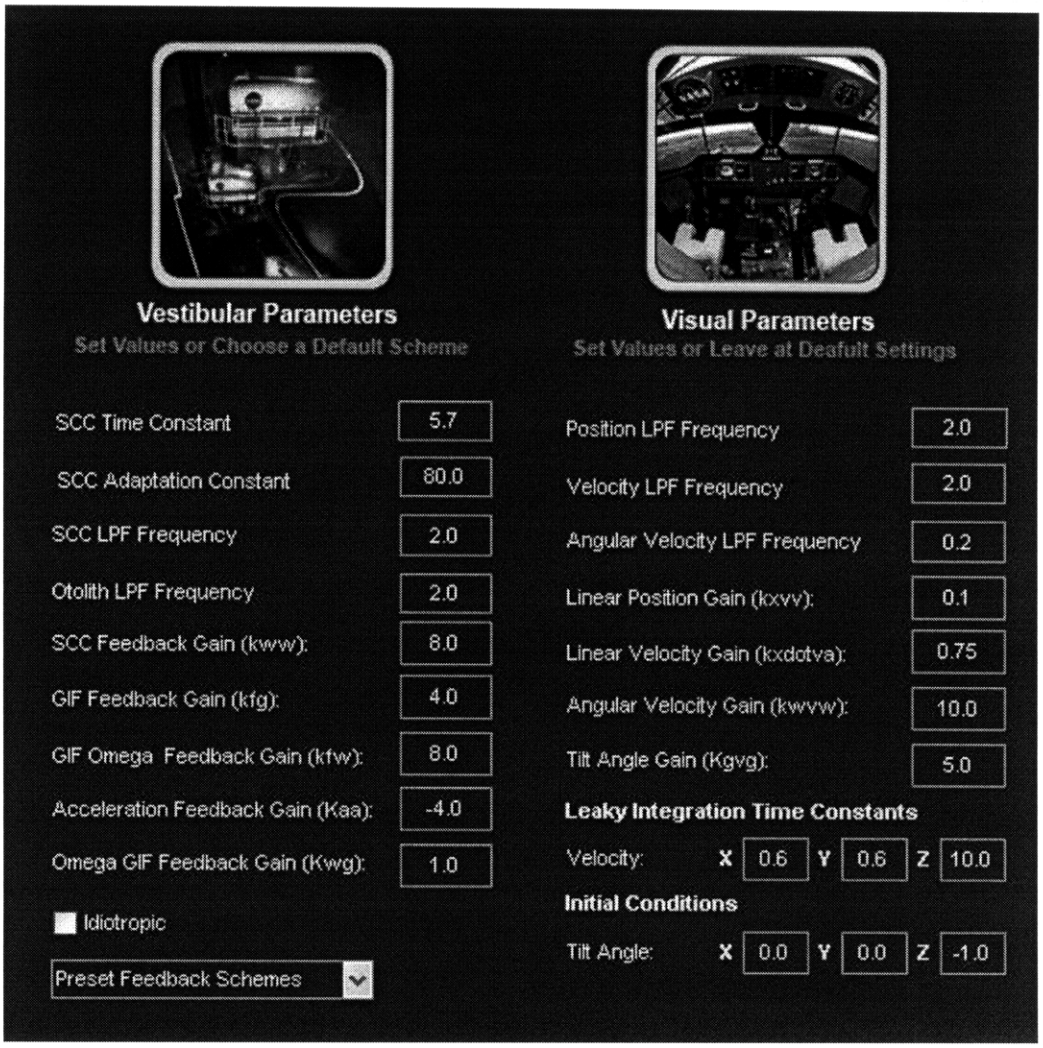
Only .xls files will load properly. If the data is correctly formatted, the Excel filename will appear in the "Selected Stimulus" box below. If not, an **"Invalid Input File"** error message box appears.

Double clicking the single dot "." on the top most line in the current directory window refreshes the current folder.

Double clicking the double dot ".." on the second line navigates to the parent directory.



Select Gravity Environment: Specify the magnitude of the gravitational field. Preset options include standard 1G Earth, 1/6 G Lunar, 3/8 G Martian and a 0 G free space environment. For additional gravity magnitudes, selecting the "1+" radio button will display a text input box allowing users to input any desired value.



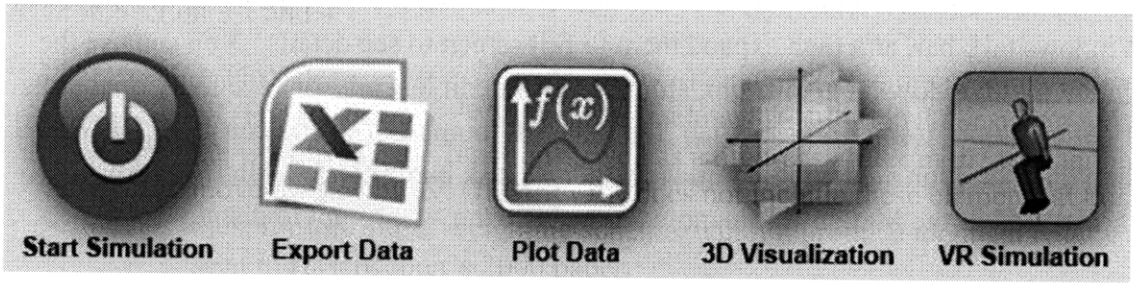
Visual Parameters: Set the visual system residual weighting parameters, the linear velocity leaky integration time constant, and the low pass filter frequencies. Preset values have been validated against the data sets considered by Borah (1979).

Vestibular Parameters: Set the Semicircular Canal & Otolith, residual weighting parameters, and Filter parameters. Preset feedback schemes allow users to adjust the weighting parameters in accordance with past and current research, as well as adjust for various species (monkey/human). The current schemes in OBSERVER V1.2 are listed below:

Note: The input values for leak rates for the acceleration to velocity leaky integrator are frequencies and not time constants. Thus these values correspond to the inverse of the values given in Section 3.1 and 4.1.

1. Haselwanter 2000 (Human) – Kw = 1.0, Kf = 10.0, Kwf = 1.0, Ka = -1.0, Kfw = 1.0
    a. NOTE: Haselwanter's 2001 model included an additional constant z-axis force in the Otolith transfer function as well as a high-pass filter in the translational eye movement pathway. OBSERVER does not include these elements at this time and thus, for this preset weighting scheme, will not produce identical results to those seen in Haselwanter's 2000 paper.
2. Merfeld 1993 (Monkey) – Kw = 3.0, Kf = 2.0, Kwf = 20.0, Ka = -0.9, Kfw = 1.0
3. Merfeld 2002 (Human) – Kw = 3.0, Kf = 2.0, Kwf = 2.0, Ka = -2.0, Kfw = 1.0
4. Merfeld 2002 (Monkey) –Kw = 5.0, Kf = 10.0, Kwf = 100.0, Ka = -5.0, Kfw = 1.0
5. Vingerhoets 2007 (Human) –Kw = 8.0, Kf = 4.0, Kwf = 8.0, Ka = -4.0, Kfw = 1.0
    a. NOTE: Vingerhoets 2007 model utilizes a similar Idiotropic bias to that incorporated in OBSERVER. For constancy with Vingerhoets results it is suggested that the Idiotropic Bias sensory cue be toggled on and set to a value of 0.2. In addition Vingerhoets included a Leaky Integration to estimate translation velocity with a time constant of 0.06 sec.

Control Panel:



Push the **Start Stimulation** button to read the model parameters set on the panels, and run the simulation. Calculations are complete when the "Plot Data," "Export Data," "3D Visualization," and "VR Simulation" buttons become active.

Push the **Export Data** button to export all data to an excel spreadsheet. The spreadsheet will appear in the current directory with file name "Observer Data Export-XX-XX-XX-X-XX" where the sequence of X's corresponds to the export date and time.

Push the **Plot Data** button to display results vs. time in labeled windows:

ACC – Actual and estimated linear acceleration

OMEGA – Actual and estimated angular velocity

G – Actual gravity vector and estimated gravity vector

GIF – Actual gravitoinertial force and estimated gravitoinertial force stimulus to the otoliths. GIF = gravity minus linear acceleration, the physical quantity tracked by a pendulum.

DISP – Actual linear displacement and estimated linear displacement

LIN VEL – Actual linear velocity and estimated linear velocity

VOR – Translational VOR and VOR (combined translational and angular)

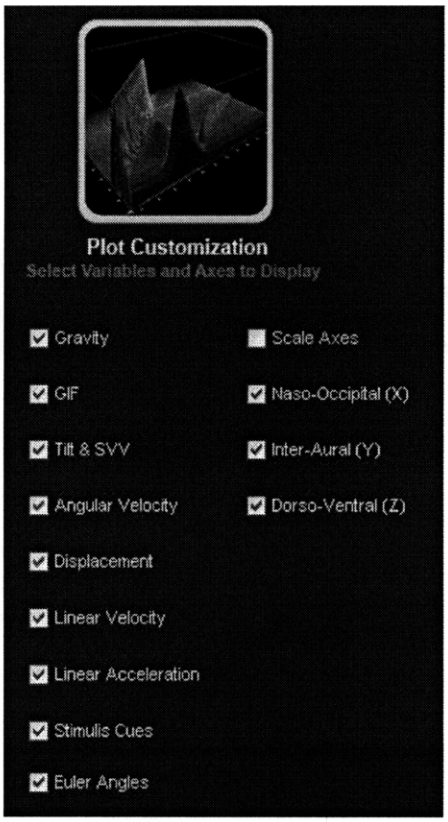
SVV – Tilt angle and subjective visual vertical

EULER – Actual and perceived roll, pitch and yaw angles

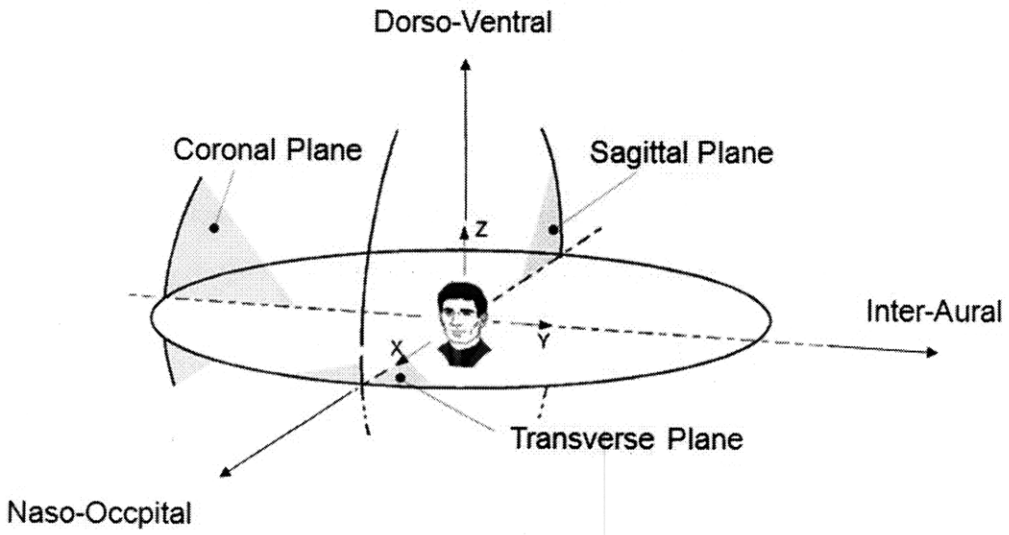
STIMULUS - Visual stimulus cues ON/OFF during stimulus

The easiest way to view individual OBSERVER figures is to call the forward using the buttons on your Windows task bar, and then expand them to full screen to see details. You can use the generic Matlab plot window tools to zoom, annotate, print, edit legends, or save individual plots in a variety of common formats (e.g. Matlab .fig, .jpg, .tif, .bmp, .pdf). You can close individual figures using the x box (or right clicking on the windows task bar button). Alternatively, you can enter "close all" in the Matlab command window after the >> prompt to dismiss all figures.

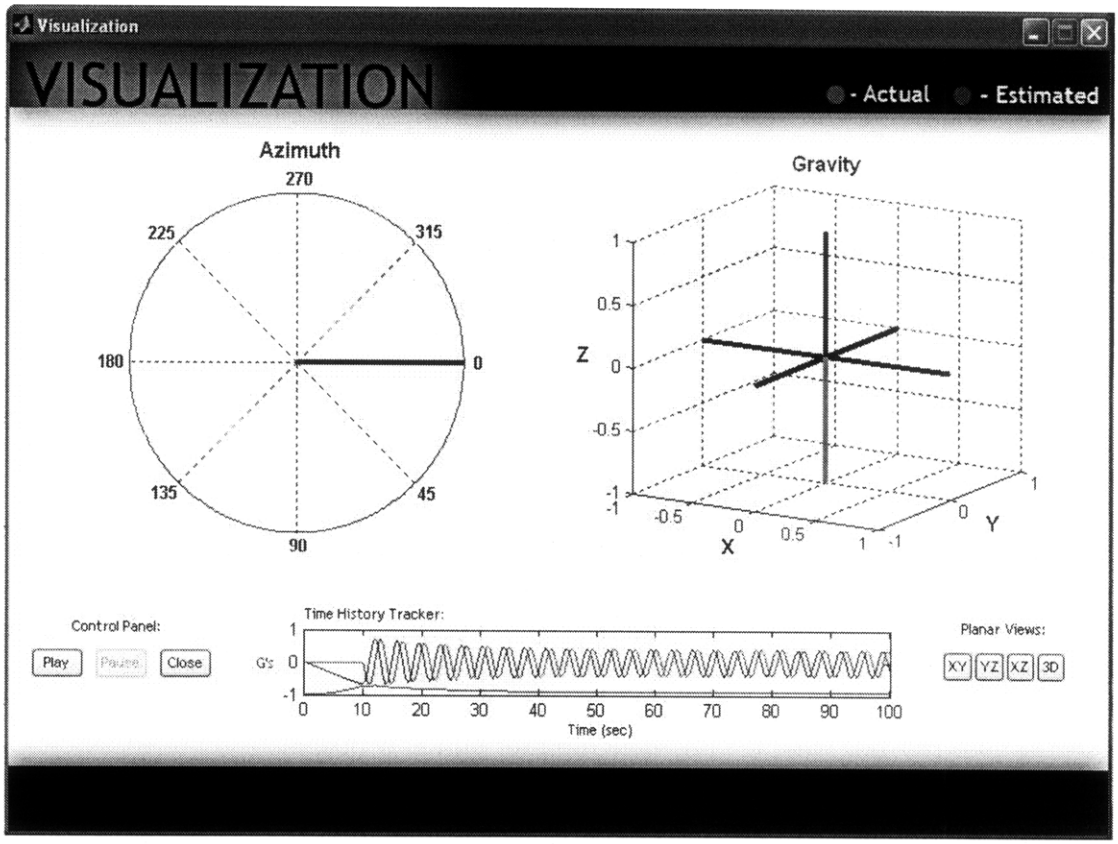
Plot Customization



1.) Individual Plot Selection – Select which variables you wish to plot when the **Plot** button is pushed.
2.) Individual Body Axis Selection – Select which axis, Naso-Occipital (X), Inter-Aural (Y), Dorso-Ventral (Z) you wish to plot when the **Plot** button is pushed.

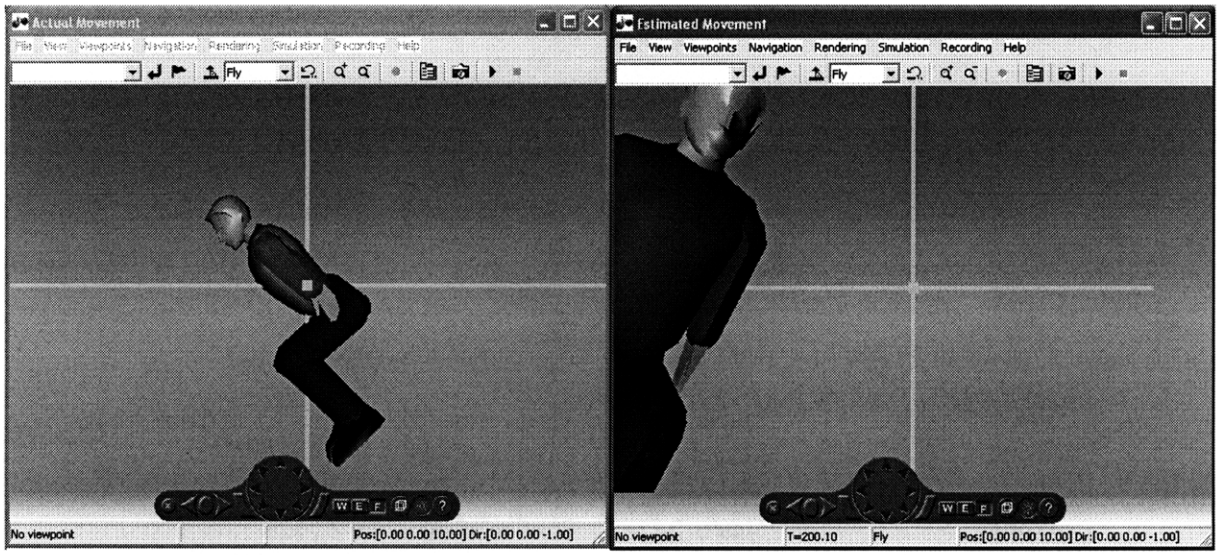


3.) Plot Scaling – Scales the actual and estimated response axis for each selected plot.
Push the **3D Visualization** button to open an animated vector plot of actual (red) vs. estimated (blue) direction of G and Azimuth vs. time. OBSERVER describes all head motion using a right handed, head fixed inertial coordinate frame, with the X axis pointing forward, the positive Y axis pointing out the left ear, and the Z axis pointing upwards as noted in the figure above.



The **Play, Pause,** and **Close** buttons start, stop or exit the animation. Always use the **Close** button provided to dismiss the visualization window. (Clicking the Windows "x" close box in the upper right hand corner creates a program error.)

Push the **VR Simulation** button to load a virtual reality simulation of a seated human subject for both the estimated and actual motion. If you do not have the VR toolbox installed you will receive an initial error while loading OBSERVER and will be unable to see the **VR** button.



When finished, you can:

1. Load a different .xls data file and run OBSERVER again.
2. Close the OBSERVER control window by clicking on the "close" menu item located in the upper right hand corner, and closing MATLAB with CTRL-Q.

Input File Format:

OBSERVER stimulus data files must be in the format shown below.

Time and Vestibular Input:

	A	B	C	D	E	F	G
1	Time	xvest	yvest	zvest	wxvest	wyvest	wzvest
2	0	0	0	0	0	0	57.2958
3	0.01	0	0	0	0	0	57.2958
4	0.02	0	0	0	0	0	57.2958
5	0.03	0	0	0	0	0	57.2958
6	0.04	0	0	0	0	0	57.2958
7	0.05	0	0	0	0	0	57.2958
8	0.06	0	0	0	0	0	57.2958
9	0.07	0	0	0	0	0	57.2958

Variable Name	Coordinate Frame	Unit	Description of vestibular input
Time	N/A	Seconds	Time vector denotes same rate of input
xvest	World	Meters	Position of subject in X_W axis
yvest	World	Meters	Position of subject in Y_W axis
zvest	World	Meters	Position of subject in Z_W axis
wxvest	Head	Deg/s	Angular Velocity of subject in X_H axis
wyvest	Head	Deg/s	Angular Velocity of subject in Y_H axis
wzvest	Head	Deg/s	Angular Velocity of subject in Z_H axis

Visual Input:

H	I	J	K	L	M	N	O	P	Q	R	S
xvis	yvis	zvis	xdotvis	ydotvis	zdotvis	wxvis	wyvis	wzvis	g_xvis	g_yvis	g_zvis
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1
0	0	0	0	0	0	0	0	-57.2958	0	0	-1

Variable Name	Coordinate Frame	Unit	Description of visual input
xvis	World	Meters	Position of subject in X_W axis
yvis	World	Meters	Position of subject in Y_W axis
zvis	World	Meters	Position of subject in Z_W axis
xdotvis	World	Meters/s	Velocity of subject in X_W axis
ydotvis	World	Meters/s	Velocity of subject in Y_W axis
zdotvis	World	Meters/s	Velocity of subject in Z_W axis
wxvis	Head	Deg/s	Angular Velocity of subject in X_H axis
wyvis	Head	Deg/s	Angular Velocity of subject in Y_H axis
wzvis	Head	Deg/s	Angular Velocity of subject in Z_H axis
g_xvis	World	N/A	X Component of "down" dir. in X_W axis
g_yvis	World	N/A	Y Component of "down" dir. in Y_W axis
g_zvis	World	N/A	Z Component of "down" dir. in Z_W axis

Variable Gravity

T
g
1
1
1
1
1
1
1
1

Variable Name	Coordinate Frame	Unit	Description
g	N/A	g's	Magnitude of gravity vector

Visual Switches

U	V	W	X
Pos ON	Vel ON	AngVel ON	G ON
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0

Variable Name	Coordinate Frame	Unit	Description
Pos ON	N/A		Boolean Visual position cue ON = 1, OFF = 0
Vel ON	N/A		Boolean Visual velocity cue ON = 1, OFF = 0
AngVel ON	N/A		Boolean Visual angular velocity ON=1, OFF=0
G ON	N/A		Boolean Visual gravity cue ON = 1, OFF = 0

Sponsoring Research Project:

NSBRI Project SA1302. "Modeling and mitigating spatial disorientation in low-gravity environments", Principal Investigator: R.L. Small, Alion Science and Technology Corporation, Boulder, CO. MIT Co-Investigators: CM Oman and LR Young.

APPENDIX C. OBSERVER MODEL (OBSERVER.M) MATLAB CODE

NOTE: The OBSERVER Spatial Orientation Analysis Tool requires MATLAB version 2007A or 2007B (or later) with Virtual Reality Toolbox version 4.5 (or later) and VR editor and VR viewer installed (for VR simulations).

```
% ****** COPYRIGHT MASSACHUSETTES INSTITUTE OF TECHNOLOGY (c) 2009 **********
% Contact: Charles Oman (COMAN@MIT.EDU) or Michael Newman (M_NEWMAN@MIT.EDU
% ********************************************************************************

% ******************* Begin initialization code - DO NOT EDIT***************
function varargout = observer(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @observer_OpeningFcn, ...
                   'gui_OutputFcn',  @observer_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%********************************************************************************

% **************Executes just before observer is made visible****************
function observer_OpeningFcn(hObject, eventdata, handles, varargin)

% Check VR Toolbox Installation
checkVR = vrinstall('-check','viewer');
if(checkVR == 1)
    %If Installed make VR Bustton Viewable
    set(handles.VR_pushbutton,'Enable','on');
    set(handles.VR_pushbutton,'CData',imread('vrON.jpg'));
else
    %If not Make Button Invisible and Display Warning Dialog
    set(handles.VR_pushbutton,'Enable','inactive');
    set(handles.VR_pushbutton,'CData',imread('vrOFF.jpg'));
    err = errordlg('You do not have VR TOOLBOX Viewer Installed Some Features
on the GUI will be disabled. If you have purchased VR TOOLBOX type
"vrinstall" in the Command Window to install the viewer and reload the
GUI.','Observer Warning!','modal');
    figure(err);
    uiwait(err);
end

% Load GUI and turn off warnings
warning('off', 'all');
```

```
movegui('center');

% Set background image and properties
ha = axes('units','normalized','position',[0 0 1 1]);
uistack(ha,'bottom');
Image = imread('back2.jpg');
plot_background = imagesc(Image);
set(ha,'handlevisibility','off','visible','off')

% Set Button Pictures as Inactive
set(handles.start_simulation,'CData',imread('startOFF.jpg'));
set(handles.export_Data,'CData',imread('exportOFF.jpg'));
set(handles.plot_pushbutton,'CData',imread('plotOFF.jpg'));
set(handles.gif_visual_pushbutton,'CData',imread('3dOFF.jpg'));
set(handles.VR_pushbutton,'CData',imread('vrOFF.jpg'));

%************************** GAIN VALUES************************************
%Values to Load when GUI starts
opening_kw = '8.0';
set(handles.kw_edit_text,'String',opening_kw);
opening_kf = '4.0';
set(handles.kf_edit_text,'String',opening_kf);
opening_kfw = '8.0';
set(handles.kfw_edit_text,'String',opening_kfw);
opening_ka = '-4.0';
set(handles.ka_edit_text,'String',opening_ka);
opening_kwf = '1.0';
set(handles.kwf_edit_text,'String',opening_kwf);
opening_kvg = '5.0';
set(handles.tilt_visual_edit_text,'String',opening_kvg);
opening_kvw = '10.0';
set(handles.omega_visual_edit_text,'String',opening_kvw);
opening_kxdotva = '0.75';
set(handles.velocity_visual_edit_text,'String',opening_kxdotva);
opening_kxvv = '0.1';
set(handles.position_visual_edit_text,'String',opening_kxvv);
opening_X_LPF = '2.0';
set(handles.position_LPF_visual_edit_text,'String',opening_X_LPF);
opening_Xdot_LPF = '2.0';
set(handles.velocity_LPF_visual_edit_text,'String',opening_Xdot_LPF);
opening_Omega_LPF = '0.2';
set(handles.omega_LPF_visual_edit_text,'String',opening_Omega_LPF);
opening_leak_x = '0.6';
set(handles.leak_x_edit_text,'String',opening_leak_x);
opening_leak_y = '0.6';
set(handles.leak_y_edit_text,'String',opening_leak_y);
opening_leak_z = '10.0';
set(handles.leak_z_edit_text,'String',opening_leak_z);
%************************************************************************

%*************** Preload Initial  Button Selections ***********************
%Gravity Set to 1G
handles.g = 1.0;

%Idiotropic SVV w*h value set to 0
w = 0;
```

```
handles.w = w;
assignin('base', 'w', w);
% Choose default command line output for observer
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
%*************************************************************************

% ************** Select Directory to Load Data Files ************************
if nargin == 3,
    initial_dir = pwd;
    initial_dir_acc = pwd;
elseif nargin > 4
    if strcmpi(varargin{1},'dir')
        if exist(varargin{2},'dir')
            initial_dir = varargin{2};
            initial_dir_acc = varargin{2};
        else
            errordlg({'Input argument must be a valid','directory'},'Input
Argument Error!')
            return
        end
    else
        errordlg('Unrecognized input argument','Input Argument Error!');
        return;
    end
end
% Populate the listbox's
load_listbox(initial_dir,handles)
%*************************************************************************



% ************ Create the Output VARARGOUT Function DO NOT EDIT **************
function varargout = observer_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%
*************************************************************************



% ********************** All Preset Feedback Schemes ************************
function feedback_popup_Callback(hObject, eventdata, handles)

val = get(hObject,'Value');
switch val
case 1
% User selected the first item
case 2
    %Human Preset Values (From Haselwanter 2000)
    handles.haselwanter_human_00_kw = '1.0';
    handles.haselwanter_human_00_kf = '10.0';
    handles.haselwanter_human_00_kfw = '1.0';
    handles.haselwanter_human_00_ka = '-1.0';
```

```
    handles.haselwanter_human_00_kwf = '1.0';
    set(handles.kw_edit_text,'String',handles.haselwanter_human_00_kw);
    set(handles.kf_edit_text,'String',handles.haselwanter_human_00_kf);
    set(handles.kfw_edit_text,'String',handles.haselwanter_human_00_kfw);
    set(handles.ka_edit_text,'String',handles.haselwanter_human_00_ka);
    set(handles.kwf_edit_text,'String',handles.haselwanter_human_00_kwf);
case 3
    %Monkey Preset Values (From Merfeld 1993)
    handles.merfeld_monkey_93_kw = '3.0';
    handles.merfeld_monkey_93_kf = '2.0';
    handles.merfeld_monkey_93_kfw = '20';
    handles.merfeld_monkey_93_ka = '-0.9';
    handles.merfeld_monkey_93_kwf = '1.0';
    set(handles.kw_edit_text,'String',handles.merfeld_monkey_93_kw);
    set(handles.kf_edit_text,'String',handles.merfeld_monkey_93_kf);
    set(handles.kfw_edit_text,'String',handles.merfeld_monkey_93_kfw);
    set(handles.ka_edit_text,'String',handles.merfeld_monkey_93_ka);
    set(handles.kwf_edit_text,'String',handles.merfeld_monkey_93_kwf);

case 4
    %Human Preset Values (From Merfeld, Zupan 2002)
    handles.merfeld_human_02_kw = '3.0';
    handles.merfeld_human_02_kf = '2.0';
    handles.merfeld_human_02_kfw = '2.0';
    handles.merfeld_human_02_ka = '-2.0';
    handles.merfeld_human_02_kwf = '1.0';
    set(handles.kw_edit_text,'String',handles.merfeld_human_02_kw);
    set(handles.kf_edit_text,'String',handles.merfeld_human_02_kf);
    set(handles.kfw_edit_text,'String',handles.merfeld_human_02_kfw);
    set(handles.ka_edit_text,'String',handles.merfeld_human_02_ka);
    set(handles.kwf_edit_text,'String',handles.merfeld_human_02_kwf);
case 5
    %Monkey Preset Values (From Merfeld, Zupan 2002)
    handles.merfeld_monkey_02_kw = '5.0';
    handles.merfeld_monkey_02_kf = '10.0';
    handles.merfeld_monkey_02_kfw = '100.0';
    handles.merfeld_monkey_02_ka = '-5.0';
    handles.merfeld_monkey_02_kwf = '1.0';
    set(handles.kw_edit_text,'String',handles.merfeld_monkey_02_kw);
    set(handles.kf_edit_text,'String',handles.merfeld_monkey_02_kf);
    set(handles.kfw_edit_text,'String',handles.merfeld_monkey_02_kfw);
    set(handles.ka_edit_text,'String',handles.merfeld_monkey_02_ka);
    set(handles.kwf_edit_text,'String',handles.merfeld_monkey_02_kwf);
case 6
    %Human Preset Values (From Vingerhoets 2007)
    handles.vingerhoets_human_07_kw = '8.0';
    handles.vingerhoets_human_07_kf = '4.0';
    handles.vingerhoets_human_07_kfw = '8.0';
    handles.vingerhoets_human_07_ka = '-4.0';
    handles.vingerhoets_human_07_kwf = '1.0';
    set(handles.kw_edit_text,'String',handles.vingerhoets_human_07_kw);
    set(handles.kf_edit_text,'String',handles.vingerhoets_human_07_kf);
    set(handles.kfw_edit_text,'String',handles.vingerhoets_human_07_kfw);
    set(handles.ka_edit_text,'String',handles.vingerhoets_human_07_ka);
    set(handles.kwf_edit_text,'String',handles.vingerhoets_human_07_kwf);
end
guidata(hObject,handles);
```

```
%*************************************************************************

% ************Gravity Environment Selection Buttons ************************
function gravity_panel_SelectionChangeFcn(hObject, eventdata, handles)

% Gravity Switch Box
switch get(hObject,'Tag')    % Get Tag of selected object
    case 'radio_1g'
        set([handles.hyper_edit_text],'Visible','off');
        handles.g = 1.0;
    case 'radio_38g'
        set([handles.hyper_edit_text],'Visible','off');
        handles.g = 3/8.;
    case 'radio_16g'
        set([handles.hyper_edit_text],'Visible','off');
        handles.g = 1/6.;
    case 'radio_0g'
        set([handles.hyper_edit_text],'Visible','off');
        handles.g = 0;
    case 'radio_hyper'
        set([handles.hyper_edit_text],'Visible','on');
        set([handles.hyper_edit_text],'String','1.0');


end
guidata(hObject,handles);
%*************************************************************************



% ********************Idiotropic Logic Switch ******************************
function idiot_checkbox_Callback(hObject, eventdata, handles)

% Idiotropic Check Box and Popup Edit Text Box
if (get(hObject,'Value') == get(hObject,'Max'))
    %Idiotropic Bias
    set([handles.w_edit_text],'Visible','on');
    set([handles.w_edit_text],'String','0.0');
else
    %Idiotropic
    set([handles.w_edit_text],'Visible','off');
    set([handles.w_edit_text],'String','0.0');
end
guidata(hObject,handles);
%*************************************************************************



% ************ Load Angular Velocity Data File List Box **********************
function load_listbox(dir_path, handles)
cd (dir_path)
dir_struct = dir(dir_path);
[sorted_names,sorted_index] = sortrows({dir_struct.name}');
handles.file_names = sorted_names;
handles.is_dir = [dir_struct.isdir];
```

```
handles.sorted_index = sorted_index;
guidata(handles.figure1,handles)
set(handles.import_list_box,'String',handles.file_names,...
    'Value',1)
set(handles.import_text,'String',pwd)
%**************************************************************************



% *List Box File/Dir Selection and Creation/Deletion of additional directory
Paths *
function import_list_box_Callback(hObject, eventdata, handles)

get(handles.figure1,'SelectionType');
% If double click
if strcmp(get(handles.figure1,'SelectionType'),'open')
    index_selected = get(handles.import_list_box,'Value');
    file_list = get(handles.import_list_box,'String');
    % Item selected in list box
    filename = file_list{index_selected};
    % If directory
    if  handles.is_dir(handles.sorted_index(index_selected))
        cd (filename)
        % Load list box with new directory.
        load_listbox(pwd,handles)
    else
        [path,name,ext,ver] = fileparts(filename);
        switch ext
            case '.xls'
                % Load in Data File and Change Text
                set(handles.loaded_file_text,'String',[name, '.xls']);
                handles.file_to_open = [path,name,ext];
                assignin('base', 'file', handles.file_to_open);
                set([handles.start_simulation],'Enable','on')
                set([handles.duration_text],'Visible','off');
                set([handles.sim_Time_text],'Visible','off');
                set([handles.sample_rate_text],'Visible','off');
                set([handles.plot_pushbutton],'Enable','inactive');
                set([handles.gif_visual_pushbutton],'Enable','inactive');
                set([handles.export_Data],'Enable','inactive');
                set([handles.VR_pushbutton],'Enable','inactive');
                set(handles.start_simulation,'CData',imread('startON.jpg'));
                set(handles.export_Data,'CData',imread('exportOFF.jpg'));
                set(handles.plot_pushbutton,'CData',imread('plotOFF.jpg'));

set(handles.gif_visual_pushbutton,'CData',imread('3dOFF.jpg'));
                set(handles.VR_pushbutton,'CData',imread('vrOFF.jpg'));
                guidata(hObject,handles);
            otherwise
                errordlg('Data file must be of .xls extension.', 'Invalid
Input File')
        end
    end
end

function import_list_box_CreateFcn(hObject, eventdata, handles)
```

```
usewhitebg = 0;
if usewhitebg
    set(hObject,'BackgroundColor','white');
else
    %set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function figure1_CreateFcn(hObject, eventdata, handles)

setappdata(hObject, 'StartPath', pwd);
addpath(pwd);

function figure1_DeleteFcn(hObject, eventdata, handles)

if isappdata(hObject, 'StartPath')
    rmpath(getappdata(hObject, 'StartPath'));
end
%*************************************************************************



%****************** LOAD VALUES AND START SIMULATION ************************
function start_simulation_Callback(hObject, eventdata, handles)

%Disable buttons
set([handles.plot_pushbutton],'Enable','inactive');
set([handles.gif_visual_pushbutton],'Enable','inactive');
set([handles.start_simulation],'Enable','inactive');
set([handles.export_Data],'Enable','inactive');
set([handles.VR_pushbutton],'Enable','inactive');
set(handles.start_simulation,'CData',imread('startOFF.jpg'));
set(handles.export_Data,'CData',imread('exportOFF.jpg'));
set(handles.plot_pushbutton,'CData',imread('plotOFF.jpg'));
set(handles.gif_visual_pushbutton,'CData',imread('3dOFF.jpg'));
set(handles.VR_pushbutton,'CData',imread('vrOFF.jpg'));


%Start progress bar and check if gravity state is a predefined quantity or
%if the user inputed its own value in the edit text box.
pause(0.5);
progress = waitbar(0,'Please wait...');

if strcmp(get([handles.hyper_edit_text],'Visible'),'on')
    handles.g = str2double(get(handles.hyper_edit_text, 'String'));
end


% Initial Conditions

%Initialize the GRAVITY input to the model [gx0 gy0 gz0]'
G0=[0 0 -handles.g]';
assignin('base', 'G0', G0);
```

```
% Initialize the internal GRAVITY STATE of the model
GG0=G0;
assignin('base', 'GG0', GG0);

% Tilt Angle
g_x = str2double(get(handles.x_tilt_IC, 'String'));
g_y = str2double(get(handles.y_tilt_IC, 'String'));
g_z = str2double(get(handles.z_tilt_IC, 'String'));
g_mag = sqrt(g_x*g_x + g_y*g_y + g_z*g_z);
g_norm = [g_x/g_mag g_y/g_mag g_z/g_mag]';
assignin('base', 'g_norm', g_norm);

% Initialize Quaternions
if g_norm(1) == G0(1) && g_norm(2) == G0(2)
    Q0 = [1 0 0 0]';
    VR_IC = [0 0 0 0];
else
    % Perpendicular Vector
    E_vec = CROSS(g_norm,[0 0 -1]);
    % Normalize E vector
    E_mag = sqrt(E_vec(1)*E_vec(1) + E_vec(2)*E_vec(2) + E_vec(3)*E_vec(3));
    E = E_vec./E_mag;
    % Calculate Rotation angle
    E_angle = acos(DOT(g_norm,[0 0 -1]));
    % Calculate Quaternion
    Q0 = [cos(E_angle/2) E(1)*sin(E_angle/2) E(2)*sin(E_angle/2)
E(3)*sin(E_angle/2)]';
    VR_IC = [E,E_angle];
end

assignin('base', 'Q0', Q0);
assignin('base', 'VR_IC', VR_IC);


% Preload Idiotropic Vecdtor
h = [0 0 -1];
assignin('base', 'h', h);

% Preload Idiotropic Bias
w = 0;
assignin('base', 'w', w);

% Initialize scc time constants [x y z]'
handles.tau_scc_value = str2double(get(handles.tau_scc_edit_text, 'String'));
tau_scc=handles.tau_scc_value*[1 1 1]';
assignin('base', 'tau_scc', tau_scc);

%Internal Model SCC Time Constant is Set to CNS time constant,
tau_scc_cap=tau_scc;
assignin('base', 'tau_scc_cap', tau_scc_cap);

% Initialize scc adaptation time constants
handles.tau_a_value = str2double(get(handles.tau_a_edit_text, 'String'));
tau_a=handles.tau_a_value*[1 1 1]';
assignin('base', 'tau_a', tau_a);
```

```
% Initialize the low-pass filter frequency for scc
handles.f_oto = str2double(get(handles.f_oto_edit_text, 'String'));
f_oto=handles.f_oto;
assignin('base', 'f_oto', f_oto);

% Initialize the lpf frequency for otolith
handles.f_scc = str2double(get(handles.f_scc_edit_text, 'String'));
f_scc=handles.f_scc;
assignin('base', 'f_scc', f_scc);

% Initialize the Ideotropic Bias Amount 'w'
handles.w = str2double(get(handles.w_edit_text, 'String'));
w=handles.w;
assignin('base', 'w', w);

% Initialize Kww feedback gain
handles.kww = str2double(get(handles.kw_edit_text,'String'))*[1 1 1]';
assignin('base', 'kww', handles.kww);
guidata(hObject,handles);

% Initialize Kfg feedback gain
handles.kfg = str2double(get(handles.kf_edit_text,'String'))*[1 1 1]';
assignin('base', 'kfg', handles.kfg);
guidata(hObject,handles);

% Initialize Kfw feedback gain
handles.kfw = str2double(get(handles.kfw_edit_text,'String'))*[1 1 1]';
assignin('base', 'kfw', handles.kfw);
guidata(hObject,handles);

% Initialize Kaa feedback gain
handles.kaa = str2double(get(handles.ka_edit_text,'String'))*[1 1 1]';
assignin('base', 'kaa', handles.kaa);
guidata(hObject,handles);

% Initialize Kwg feedback gain
handles.kwg = str2double(get(handles.kwf_edit_text,'String'))*[1 1 1]';
assignin('base', 'kwg', handles.kwg);
guidata(hObject,handles);

% Initialize Kvg feedback gain
handles.kgvg = str2double(get(handles.tilt_visual_edit_text,'String'))*[1 1
1]';
assignin('base', 'kgvg', handles.kgvg);
guidata(hObject,handles);

% Initialize Kvw feedback gain
handles.kwvw = str2double(get(handles.omega_visual_edit_text,'String'))*[1 1
1]';
assignin('base', 'kwvw', handles.kwvw);
guidata(hObject,handles);

% Initialize Kxdotva feedback gain
```

```
handles.kxdotva =
str2double(get(handles.velocity_visual_edit_text,'String'))*[1 1 1]';
assignin('base', 'kxdotva', handles.kxdotva);
guidata(hObject,handles);

% Initialize Kxvv feedback gain
handles.kxvv = str2double(get(handles.position_visual_edit_text,'String'))*[1
1 1]';
assignin('base', 'kxvv', handles.kxvv);
guidata(hObject,handles);

% Initialize Visual Position LPF Frequency
handles.f_visX =
str2double(get(handles.position_LPF_visual_edit_text,'String'));
assignin('base', 'f_visX', handles.f_visX);
guidata(hObject,handles);

% Initialize Visual Velocity LPF Frequency
handles.f_visV =
str2double(get(handles.velocity_LPF_visual_edit_text,'String'));
assignin('base', 'f_visV', handles.f_visV);
guidata(hObject,handles);

% Initialize Visual Angular Velocity LPF Frequency
handles.f_visO =
str2double(get(handles.omega_LPF_visual_edit_text,'String'));
assignin('base', 'f_visO', handles.f_visO);
guidata(hObject,handles);

% Initialize Graviceptor Gain
oto_a = 60*[1 1 1]';
assignin('base','oto_a', oto_a);
guidata(hObject,handles);

% Initialize Adapatation time constant
oto_Ka = 1.3*[1 1 1]';
assignin('base','oto_Ka', oto_Ka);
guidata(hObject,handles);

% Initialize  X Leaky Integration Time Constant
handles.x_leak = str2double(get(handles.leak_x_edit_text,'String'));
assignin('base', 'x_leak', handles.x_leak);
guidata(hObject,handles);

% Initialize  Y Leaky Integration Time Constant
handles.y_leak = str2double(get(handles.leak_y_edit_text,'String'));
assignin('base', 'y_leak', handles.y_leak);
guidata(hObject,handles);

% Initialize  X Leaky Integration Time Constant
handles.z_leak = str2double(get(handles.leak_z_edit_text,'String'));
assignin('base', 'z_leak', handles.z_leak);
guidata(hObject,handles);

%Load data file
```

```
input_file= xlsread(handles.file_to_open);
time = input_file(:,1);
x_in = [input_file(:,2),input_file(:,3),input_file(:,4)];
omega_in = [input_file(:,5),input_file(:,6),input_file(:,7)];
xv_in = [input_file(:,8),input_file(:,9),input_file(:,10)];
xvdot_in = [input_file(:,11),input_file(:,12),input_file(:,13)];
omegav_in = [input_file(:,14),input_file(:,15),input_file(:,16)];
gv_in = [input_file(:,17),input_file(:,18),input_file(:,19)];
g_variable_in =input_file(:,20);
pos_ON =input_file(:,21);
vel_ON =input_file(:,22);
angVel_ON =input_file(:,23);
g_ON =input_file(:,24);

% Time and Tolerance Properties set by data input file to ensure a correct
% sampling rate.
delta_t = time(2) - time (1);
duration = length(time)*delta_t;
handles.duration = duration;
handles.sample_rate = 1/delta_t;
guidata(hObject,handles);
t = time;
tolerance = 0.02;

% Differentiate Position to Velcoity to Acceleration
v_in = zeros(size(x_in,1),3);
v_in(1:size(x_in,1)-1,:) = diff(x_in,1)/delta_t;

a_in = zeros(size(x_in,1),3);
a_in(1:size(x_in,1)-2,:) = diff(x_in,2)/(delta_t*delta_t);

% If we want to input straight acceleration NEED TO COMMENT OUT ABOVE and
% uncomment the below. Note that Actual Position and Actual velocity plots
% will be innaccurate if we do so.
%a_in = x_in;
%v_in = a_in;


% Set File Information GUI strgns
duration_string =num2str(duration);
delta_t_string =num2str(1/delta_t);
set([handles.duration_text],'Visible','on');
set([handles.sample_rate_text],'Visible','on');
set(handles.duration_text,'String',[duration_string, ' secs']);
set(handles.sample_rate_text,'String',[delta_t_string, ' HZ']);

% Read Data to Workspace
assignin('base', 't', t);
assignin('base', 'x_in', x_in);
assignin('base', 'xv_in', xv_in);
assignin('base', 'a_in', a_in);
assignin('base', 'v_in', v_in);
assignin('base', 'xvdot_in', xvdot_in);
assignin('base', 'omega_in', omega_in);
assignin('base', 'omegav_in', omegav_in);
```

```
assignin('base', 'gv_in', gv_in);
assignin('base', 'g_variable_in', g_variable_in);
assignin('base', 'pos_ON', pos_ON);
assignin('base', 'vel_ON', vel_ON);
assignin('base', 'angVel_ON', angVel_ON);
assignin('base', 'g_ON', g_ON);
assignin('base', 'delta_t', delta_t);
assignin('base', 'duration', duration);
assignin('base', 'tolerance', tolerance);
handles.t = t;
handles.x_in = x_in;

% Load the VR model or the non VR model depending on state of VR tookbox.
checkVR = vrinstall('-check','viewer');

if(checkVR == 1)
    model='observerModel';
    vrsetpref('DefaultViewer','internal');
else
    model='observerModelnoVR';
end
waitbar(0.2);
tic;
% Execute Simulink Model
options=simset('Solver','ode45','MaxStep',tolerance,'RelTol',tolerance,'AbsTo
l',tolerance);
[t_s, XDATA, a_est, gif_est, gif_head, a_head,
omega_head,g_head,g_est,omega_est,x_est,lin_vel_est,lin_vel,x] =
sim(model,duration,options,[]);

% Calculate Time of simulation
sim_Time = num2str(toc);
set(handles.sim_Time_text,'String',[sim_Time, ' secs']);
set([handles.sim_Time_text],'Visible','on');

waitbar(0.5);

% Bring variables from GUI to workspace
sim_time = t_s;
assignin('base', 'model', model);
assignin('base', 't_s', t_s);
assignin('base', 'sim_time', sim_time);
assignin('base', 'a_est', a_est);
assignin('base', 'omega_est', omega_est);
assignin('base', 'gif_est', gif_est);
assignin('base', 'gif_head', gif_head);
assignin('base', 'a_head', a_head);
assignin('base', 'omega_head', omega_head);
assignin('base', 'g_head', g_head);
assignin('base', 'g_est', g_est);
assignin('base', 'x_est', x_est);
assignin('base', 'lin_vel_est', lin_vel_est);
assignin('base', 'lin_vel', lin_vel);
assignin('base', 'x', x);

% Load Variables in the handle structure
```

```
handles.t_s = t_s;
handles.a_est = a_est;
handles.omega_est = omega_est;
handles.gif_est = gif_est;
handles.gif_head = gif_head;
handles.a_head = a_head;
handles.omega_head = omega_head;
handles.g_est = g_est;
handles.g_head = g_head;
handles.x_est = x_est;
handles.lin_vel_est = lin_vel_est;
handles.lin_vel = lin_vel;
handles.x = x;
handles.switch_xvdot = xvdot_switch;
handles.switch_xv = xv_switch;
handles.switch_omegav = omegav_switch;
handles.switch_gv = gv_switch;
handles.azimuth_head = azimuth_head;
handles.azimuth_est = azimuth_est;
handles.euler_head = euler_head;
handles.euler_est = euler_est;
handles.tilt = tilt;
handles.SVV = SVV;
handles.tilt_est = tilt_est;
handles.SVV_est = SVV_est;
handles.g_world = g_world;

%Calculate Angular Accelerations for SDAT

%Calculate the time step from simulink
sim_dt = t_s(size(t_s,1)) - t_s(size(t_s,1)-1);

omega_dot_head = zeros(size(omega_head,1),3);
omega_dot_head(1:size(omega_head,1)-1,:) = diff(omega_head,1)/sim_dt;

omega_dot_est = zeros(size(omega_est,1),3);
omega_dot_est(1:size(omega_est,1)-1,:) = diff(omega_est*180/pi,1)/sim_dt;

handles.omega_dot_head = omega_dot_head;
handles.omega_dot_est = omega_dot_est;

assignin('base', 'omega_dot_head', omega_dot_head);
assignin('base', 'omega_dot_est', omega_dot_est);
assignin('base', 'sim_dt', sim_dt);

%Calculate Vertical, SVV, Tilt, and Estimated Tilt, along with Errors
tilt_estTEMP(:,1) = tilt_est(1,1,:);
tiltTEMP(:,1) = tilt(1,1,:);
tilt = tiltTEMP;
tilt_est = tilt_estTEMP;
SVV = SVV*180/3.14159;
SVV_est = SVV_est*180/3.14159;
tilt = real(tilt*180/3.14159);
tilt_est = real(tilt_est*180/3.14159);
plot_SVV(:,1) = SVV;
```

```
plot_SVV(:,2) = SVV_est;
plot_tilt(:,1) = tilt;
plot_tilt(:,2) = tilt_est;
handles.plot_tilt = plot_tilt;
handles.plot_SVV = plot_SVV;
assignin('base', 'plot_tilt', plot_tilt);
assignin('base', 'plot_SVV', plot_SVV);
assignin('base', 'azimuth_est', azimuth_est);
assignin('base', 'azimuth_head', azimuth_head);
assignin('base', 'euler_head', euler_head);
assignin('base', 'euler_est', euler_est);
assignin('base', 'switch_xvdot', handles.switch_xvdot(:,1));
assignin('base', 'switch_xv', handles.switch_xv(:,1));
assignin('base', 'switch_omegav', handles.switch_omegav(:,1));
assignin('base', 'switch_gv', handles.switch_gv(:,1));
assignin('base', 'tilt_est', tilt_est);
assignin('base', 'SVV_est', SVV_est);
assignin('base', 'tilt', tilt);
assignin('base', 'SVV', SVV);
assignin('base', 'g_world', g_world);

waitbar(1.0);
close(progress)

% Re-enable all buttons
set([handles.plot_pushbutton],'Enable','on');
set([handles.export_Data],'Enable','on');
set([handles.gif_visual_pushbutton],'Enable','on');
set([handles.start_simulation],'Enable','on');
set([handles.VR_pushbutton],'Enable','on');
set(handles.start_simulation,'CData',imread('startON.jpg'));
set(handles.export_Data,'CData',imread('exportON.jpg'));
set(handles.plot_pushbutton,'CData',imread('plotON.jpg'));
set(handles.gif_visual_pushbutton,'CData',imread('3dON.jpg'));
set(handles.VR_pushbutton,'CData',imread('vrON.jpg'));
guidata(hObject,handles);
%*************************************************************************



% ************************** MENU ITEMS ************************************
function menu_close_Callback(hObject, eventdata, handles)

close(handles.figure1);

function menu_help_Callback(hObject, eventdata, handles)

HelpPath = which('help.htm');
  web(HelpPath);

function menu_about_Callback(hObject, eventdata, handles)

HelpPath = which('About.html');
  web(HelpPath, '-noaddressbox');
%*************************************************************************
```

```
% **********Visualization, VR and Signal Builder Buttons ******************
function gif_visual_pushbutton_Callback(hObject, eventdata, handles)
plotToolsAzi;

function VR_pushbutton_Callback(hObject, eventdata, handles)
load_system('observerModel')
load_system('observerModel/VR Visualization (actual)')
open_system('observerModel/VR Visualization (actual)/Actual Movement')
close_system('observerModel/VR Visualization (actual)')
open_system('observerModel/VR Visualization (estimated)/Estimated Motion')
load_system('observerModel/VR Visualization (estimated)')
close_system('observerModel/VR Visualization (estimated)')
%*************************************************************************



% ******************************** PLOTTING *********************************
function plot_pushbutton_Callback(hObject, eventdata, handles)

if get(handles.gif_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t_s),1);
     to_plot_est = zeros(length(handles.t_s),1);

    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.gif_head(:,1);
        to_plot_est(:,num_col) = handles.gif_est(:,1);
        x_on = 1;
    end

    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.gif_head(:,2);
        to_plot_est(:,num_col) = handles.gif_est(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.gif_head(:,3);
        to_plot_est(:,num_col) = handles.gif_est(:,3);
        z_on = 1;
    end
```

```
figure1 = figure('Name','GIF','NumberTitle','on');

% Create subplot 1
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('GIF','FontSize',14);

plot1 = plot(handles.t_s,to_plot_act,'LineWidth',2);

if x_on == 1
    set(plot1(1),'DisplayName','G_x');
    if y_on == 1
        set(plot1(2),'DisplayName','G_y');
        if z_on == 1
            set(plot1(3),'DisplayName','G_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','G_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','G_y');
```

```
        if z_on == 1
            set(plot1(2),'DisplayName','G_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','G_z');
        end
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end

box('on');
hold('all');


title('Estimated GIF','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','G_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','G_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','G_zest');
        end
    else
        if z_on == 1
            set(plot2(2),'DisplayName','G_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','G_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','G_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','G_zest');
        end
```

```
    end
end

xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);


legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end
```

```
if get(handles.lin_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t_s),1);
     to_plot_est = zeros(length(handles.t_s),1);

    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.a_head(:,1);
        to_plot_est(:,num_col) = handles.a_est(:,1);
        x_on = 1;
    end

    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.a_head(:,2);
        to_plot_est(:,num_col) = handles.a_est(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.a_head(:,3);
        to_plot_est(:,num_col) = handles.a_est(:,3);
        z_on = 1;
    end
```

```
figure1 = figure('Name','ACC','NumberTitle','on');

% Create subplot 1
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('Acceleration','FontSize',14);

plot1 = plot(handles.t_s,to_plot_act,'LineWidth',2);

if x_on == 1
    set(plot1(1),'DisplayName','A_x');
    if y_on == 1
        set(plot1(2),'DisplayName','A_y');
        if z_on == 1
            set(plot1(3),'DisplayName','A_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','A_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','A_y');
```

```
        if z_on == 1
            set(plot1(2),'DisplayName','A_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','A_z');
        end
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end

box('on');
hold('all');


title('Estimated Acceleration','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','A_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','A_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','A_zest');
        end
    else
        if z_on == 1
            set(plot2(2),'DisplayName','A_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','A_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','A_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','A_zest');
        end
```

```
    end
end

xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);


legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end
```

```
if get(handles.ang_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t_s),1);
     to_plot_est = zeros(length(handles.t_s),1);
     handles.omega_est_plot = handles.omega_est*180/pi;


    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.omega_head(:,1);
        to_plot_est(:,num_col) = handles.omega_est_plot(:,1);
        x_on = 1;
    end

    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.omega_head(:,2);
        to_plot_est(:,num_col) = handles.omega_est_plot(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.omega_head(:,3);
        to_plot_est(:,num_col) = handles.omega_est_plot(:,3);
        z_on = 1;
    end
```

```
figure1 = figure('Name','OMEGA','NumberTitle','on');

% Create subplot 1
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('Angular Velocity','FontSize',14);

plot1 = plot(handles.t_s,to_plot_act,'LineWidth',2);

if x_on == 1
    set(plot1(1),'DisplayName','Omega_x');
    if y_on == 1
        set(plot1(2),'DisplayName','Omega_y');
        if z_on == 1
            set(plot1(3),'DisplayName','Omega_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','Omega_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','Omega_y');
        if z_on == 1
```

```
            set(plot1(2),'DisplayName','Omega_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','Omega_z');
        end
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('deg/sec','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end

box('on');
hold('all');


title('Estimated Angular Velocity','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','Omega_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','Omega_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','Omega_zest');
        end
    else
        if z_on == 1
            set(plot2(2),'DisplayName','Omega_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','Omega_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','Omega_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','Omega_zest');
        end
    end
```

```
end

xlabel('Time (sec)','FontSize',12);
ylabel('deg/sec','FontSize',12);


legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end



if get(handles.gravity_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t_s),1);
     to_plot_est = zeros(length(handles.t_s),1);

    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.g_head(:,1);
        to_plot_est(:,num_col) = handles.g_est(:,1);
        x_on = 1;
    end

    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.g_head(:,2);
        to_plot_est(:,num_col) = handles.g_est(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.g_head(:,3);
        to_plot_est(:,num_col) = handles.g_est(:,3);
        z_on = 1;
    end



figure1 = figure('Name','G','NumberTitle','on');

% Create subplot 1
```

```
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('Gravity','FontSize',14);

plot1 = plot(handles.t_s,to_plot_act,'LineWidth',2);

if x_on == 1
    set(plot1(1),'DisplayName','g_x');
    if y_on == 1
        set(plot1(2),'DisplayName','g_y');
        if z_on == 1
            set(plot1(3),'DisplayName','g_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','g_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','g_y');
        if z_on == 1
            set(plot1(2),'DisplayName','g_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','g_z');
        end
```

```
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end

box('on');
hold('all');


title('Estimated Gravity','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','g_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','g_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','g_zest');
        end
    else
        if z_on == 1
            set(plot2(2),'DisplayName','g_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','g_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','g_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','g_zest');
        end
    end
end

xlabel('Time (sec)','FontSize',12);
ylabel('G''s','FontSize',12);
```

```
legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end


if get(handles.cue_plot_check,'Value')==1.0

figure1 = figure('Name','Cues','NumberTitle','on');

% Create axes
axes1 = axes('Parent',figure1,'YTickLabel',{'On','Off'},'YTick',[0 1],...
    'Position',[0.13 0.7673 0.775 0.1577],...
    'FontSize',12);
% Uncomment the following line to preserve the X-limits of the axes
xlim([0 max(handles.t_s)]);
% Uncomment the following line to preserve the Y-limits of the axes
ylim([-0.5 1.5]);
box('on');
hold('all');

% Create title
title('Visual Position Cue','FontWeight','bold','FontSize',12);

% Create plot
plot(handles.t_s,handles.switch_xv(:,1),'Parent',axes1,'LineWidth',3,'LineSty
le',':',...
    'DisplayName','switch_xv vs t_s',...
    'Color',[0 0 0]);

% Create axes
axes2 = axes('Parent',figure1,'YTickLabel',{'On','Off'},'YTick',[0 1],...
    'Position',[0.13 0.5482 0.775 0.1577],...
    'FontSize',12);
% Uncomment the following line to preserve the X-limits of the axes
xlim([0 max(handles.t_s)]);
% Uncomment the following line to preserve the Y-limits of the axes
ylim([-0.5 1.5]);
box('on');
hold('all');

% Create title
title('Visual Linear Velocity Cue','FontWeight','bold','FontSize',12);

% Create plot
plot(handles.t_s,handles.switch_xvdot(:,1),'Parent',axes2,'LineWidth',3,'Line
Style',':',...
    'Color',[0 0.498 0],...
    'DisplayName','switch_xvdot vs t_s');

% Create axes
```

```
axes3 = axes('Parent',figure1,'YTickLabel',{'On','Off'},'YTick',[0 1],...
    'Position',[0.13 0.3291 0.775 0.1577],...
    'FontSize',12);
% Uncomment the following line to preserve the X-limits of the axes
xlim([0 max(handles.t_s)]);
% Uncomment the following line to preserve the Y-limits of the axes
ylim([-0.5 1.5]);
box('on');
hold('all');

% Create title
title('Visual Angular Velocity','FontWeight','bold','FontSize',12);

% Create plot
plot(handles.t_s,handles.switch_omegav(:,1),'Parent',axes3,'LineWidth',3,'Lin
eStyle',':',...
    'Color',[1 0 0],...
    'DisplayName','switch_omegav vs t_s');

% Create subplot
subplot1 = subplot(4,1,4,'Parent',figure1,'YTickLabel',{'On','Off'},...
    'YTick',[0 1],...
    'FontSize',12);
% Uncomment the following line to preserve the X-limits of the axes
xlim([0 max(handles.t_s)]);
% Uncomment the following line to preserve the Y-limits of the axes
ylim([-0.5 1.5]);
box('on');
hold('all');

% Create title
title('Visual Tilt (Gravity) Vector','FontWeight','bold','FontSize',12);

% Create plot
plot(handles.t_s,handles.switch_gv(:,1),'Parent',subplot1,'MarkerSize',8,'Lin
eWidth',3,...
    'LineStyle',':',...
    'DisplayName','switch_gv vs t_s');

% Create xlabel
xlabel('Time(sec)','FontSize',12);


end
```

```
if get(handles.position_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t),1);
     to_plot_est = zeros(length(handles.t_s),1);

    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.x_in(:,1);
        to_plot_est(:,num_col) = handles.x_est(:,1);
        x_on = 1;
    end

    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.x_in(:,2);
        to_plot_est(:,num_col) = handles.x_est(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.x_in(:,3);
        to_plot_est(:,num_col) = handles.x_est(:,3);
        z_on = 1;
    end



figure1 = figure('Name','DISP','NumberTitle','on');

% Create subplot 1
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
```

```
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('Displacement','FontSize',14);

plot1 = plot(handles.t,to_plot_act,'LineWidth',2);

if x_on == 1
    set(plot1(1),'DisplayName','D_x');
    if y_on == 1
        set(plot1(2),'DisplayName','D_y');
        if z_on == 1
            set(plot1(3),'DisplayName','D_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','D_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','D_y');
        if z_on == 1
            set(plot1(2),'DisplayName','D_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','D_z');
        end
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('Meter''s','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end
```

```
box('on');
hold('all');


title('Estimated Displacement','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','D_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','D_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','D_zest');
        end
    else
        if z_on == 1
            set(plot2(2),'DisplayName','D_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','D_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','D_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','D_zest');
        end
    end
end

xlabel('Time (sec)','FontSize',12);
ylabel('Meter''s','FontSize',12);


legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end

if get(handles.vert_plot_check,'Value')==1.0
% Create figure
figure6 = figure('Name','TILT','NumberTitle','on');

% Create subplot
subplot1 = subplot(2,1,1,'Parent',figure6,'YGrid','on','XGrid','on',...
    'FontSize',12);
box('on');
hold('all');
```

```
% Create title
title('Verticality Perception','FontSize',14);

plot1 = plot(handles.t_s,handles.plot_SVV,'LineWidth',2);
set(plot1(2),'DisplayName','SVV','Color','r');
set(plot1(1),'DisplayName','Actual Vertical','Color','b');


% Create xlabel
xlabel('Time (sec)','FontSize',12);

% Create ylabel
ylabel('Deg','FontSize',12);

% Create subplot
subplot2 = subplot(2,1,2,'Parent',figure6,'YGrid','on','XGrid','on',...
    'FontSize',12);

box('on');
hold('all');

% Create title
title('Tilt Angle Based on Gravity','FontSize',14);


plot2 = plot(handles.t_s,handles.plot_tilt,'LineWidth',2);
set(plot2(1),'DisplayName','Tilt Angle','Color','r');
set(plot2(2),'DisplayName','Estimated Tilt Angle','Color','b');


% Create xlabel
xlabel('Time (sec)','FontSize',12);

% Create ylabel
ylabel('Deg','FontSize',12);



% Create legend
legend1 = legend(subplot1,'show');


% Create legend
legend2 = legend(subplot2,'show');

end


if get(handles.euler_plot_check,'Value')==1.0
% Create figure
```

```
figure6 = figure('Name','ANGLE','NumberTitle','on');

% Create subplot
subplot1 = subplot(3,1,1,'Parent',figure6,'YGrid','on','XGrid','on',...
    'FontSize',12);
box('on');
hold('all');


% Create title
title('Roll','FontSize',14);

plot1 =
plot(handles.t_s,[handles.euler_head(:,1),handles.euler_est(:,1)],'LineWidth'
,2);
set(plot1(1),'DisplayName','Roll Angle','Color','r');
set(plot1(2),'DisplayName','Estimated Roll Angle','Color','b');


% Create xlabel
xlabel('Time (sec)','FontSize',12);

% Create ylabel
ylabel('Deg','FontSize',12);

% Create subplot
subplot2 = subplot(3,1,2,'Parent',figure6,'YGrid','on','XGrid','on',...
    'FontSize',12);

box('on');
hold('all');

% Create title
title('Pitch','FontSize',14);


plot2 =
plot(handles.t_s,[handles.euler_head(:,2),handles.euler_est(:,2)],'LineWidth'
,2);
set(plot2(1),'DisplayName','Pitch Angle','Color','r');
set(plot2(2),'DisplayName','Estimated Pitch Angle','Color','b');


% Create xlabel
xlabel('Time (sec)','FontSize',12);

% Create ylabel
ylabel('Deg','FontSize',12);


% Create subplot
subplot3 = subplot(3,1,3,'Parent',figure6,'YGrid','on','XGrid','on',...
    'FontSize',12);
```

```
box('on');
hold('all');

% Create title
title('Yaw','FontSize',14);


plot3 =
plot(handles.t_s,[handles.euler_head(:,3),handles.euler_est(:,3)],'LineWidth'
,2);
set(plot3(1),'DisplayName','Yaw Angle','Color','r');
set(plot3(2),'DisplayName','Estimated Yaw Angle','Color','b');


% Create xlabel
xlabel('Time (sec)','FontSize',12);

% Create ylabel
ylabel('Deg','FontSize',12);

% Create legend
legend1 = legend(subplot1,'show');


% Create legend
legend2 = legend(subplot2,'show');

% Create legend
legend3 = legend(subplot3,'show');

end



if get(handles.lin_vel_plot_check,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

     num_col = 0;
     x_on = 0;
     y_on = 0;
     z_on = 0;
     to_plot_act = zeros(length(handles.t_s),1);
     to_plot_est = zeros(length(handles.t_s),1);

    if get(handles.naso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.lin_vel(:,1);
        to_plot_est(:,num_col) = handles.lin_vel_est(:,1);
        x_on = 1;
    end
```

```
    if get(handles.inter_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.lin_vel(:,2);
        to_plot_est(:,num_col) = handles.lin_vel_est(:,2);
        y_on = 1;
    end

    if get(handles.dorso_checkbox,'Value')==1.0
        num_col = num_col + 1;
        to_plot_act(:,num_col) = handles.lin_vel(:,3);
        to_plot_est(:,num_col) = handles.lin_vel_est(:,3);
        z_on = 1;
    end



figure1 = figure('Name','LINVEL','NumberTitle','on');

% Create subplot 1
subplot1 =
subplot(2,1,1,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)

    if max(max(abs(to_plot_act))) > max(max(abs(to_plot_est)))
        y_high = max(max(abs(to_plot_act))) + 0.1*max(max(abs(to_plot_act)));
        y_low = -1.0*y_high;
    else
        y_high = max(max(abs(to_plot_est))) + 0.1*max(max(abs(to_plot_est)));
        y_low = -1.0*y_high;
    end

    if y_low == y_high
        y_low = y_low - 1;
        y_high = y_high + 1;
    end

    ylim([y_low y_high]);
end



box('on');
hold('all');

title('Linear Velocity','FontSize',14);

plot1 = plot(handles.t_s,to_plot_act,'LineWidth',2);

if x_on == 1
```

```
    set(plot1(1),'DisplayName','V_x');
    if y_on == 1
        set(plot1(2),'DisplayName','V_y');
        if z_on == 1
            set(plot1(3),'DisplayName','V_z');
        end
    else
        if z_on == 1
            set(plot1(2),'DisplayName','V_z');
        end
    end
else
    if y_on == 1
        set(plot1(1),'DisplayName','V_y');
        if z_on == 1
            set(plot1(2),'DisplayName','V_z');
        end
    else
        if z_on == 1
            set(plot1(1),'DisplayName','V_z');
        end
    end
end


xlabel('Time (sec)','FontSize',12);
ylabel('m/s','FontSize',12);

% Create subplot 2
subplot2 =
subplot(2,1,2,'Parent',figure1,'YGrid','on','XGrid','on','FontSize',12);

if get(handles.scale_checkbox,'Value')==1.0 &&
(get(handles.naso_checkbox,'Value')==1.0 ||
get(handles.inter_checkbox,'Value')==1.0
||get(handles.dorso_checkbox,'Value')==1.0)
       ylim([y_low y_high]);
end

box('on');
hold('all');


title('Estimated Linear Velocity','FontSize',14);

plot2 = plot(handles.t_s,to_plot_est,'LineWidth',2);

if x_on == 1
    set(plot2(1),'DisplayName','V_xest');
    if y_on == 1
        set(plot2(2),'DisplayName','V_yest');
        if z_on == 1
            set(plot2(3),'DisplayName','V_zest');
        end
    else
```

```
        if z_on == 1
            set(plot2(2),'DisplayName','V_zest');
        end
    end
else
    if y_on == 1
        set(plot2(1),'DisplayName','V_yest');
        if z_on == 1
            set(plot2(2),'DisplayName','V_zest');
        end
    else
        if z_on == 1
            set(plot2(1),'DisplayName','V_zest');
        end
    end
end

xlabel('Time (sec)','FontSize',12);
ylabel('m/s','FontSize',12);


legend1 = legend(subplot1,'show');
set(legend1,'Position',[0.8131 0.7879 0.05682 0.123]);
legend2 = legend(subplot2,'show');
set(legend2,'Position',[0.8078 0.3143 0.07273 0.123]);

end
%*************************************************************************



% ******************************** Export Data ******************************
function export_Data_Callback(hObject, eventdata, handles)


set([handles.plot_pushbutton],'Enable','inactive');
set([handles.gif_visual_pushbutton],'Enable','inactive');
set([handles.start_simulation],'Enable','inactive');
set([handles.export_Data],'Enable','inactive');
set([handles.VR_pushbutton],'Enable','inactive');
set(handles.start_simulation,'CData',imread('startOFF.jpg'));
set(handles.export_Data,'CData',imread('exportOFF.jpg'));
set(handles.plot_pushbutton,'CData',imread('plotOFF.jpg'));
set(handles.gif_visual_pushbutton,'CData',imread('3dOFF.jpg'));
set(handles.VR_pushbutton,'CData',imread('vrOFF.jpg'));

set([handles.wait_text],'Visible','on');

pause(0.5);
warning off MATLAB:xlswrite:AddSheet;

% Create File Name
time_date = clock;
year = num2str(time_date(1));
```

```
month = num2str(time_date(2));
day = num2str(time_date(3));
hour = num2str(time_date(4));
minute = num2str(time_date(5));
second = num2str(floor(time_date(6)));
file_export = strcat('Observer Data Export-',month,'-',day,'-',year,'-
',hour,'_',minute,'_',second);

% Write Data File Details Sheet

%A Column
info_string(1,1) = {'Data File Details'};
info_string(6,1) = {'SCC and Otolith Parameters'};
info_string(13,1) = {'Vestibular Feedback Gains'};
info_string(20,1) = {'Gravity Enviroment'};
info_string(23,1) = {'Visual Parameters'};
info_string(28,1) = {'Visual Feedback Gains'};

%B Column
info_string(2,2) = {'File Name'};
info_string(3,2) = {'Duration'};
info_string(4,2) = {'Sample Rate'};
info_string(7,2) = {'SCC Time Constant'};
info_string(8,2) = {'SCC Adaptation Constant'};
info_string(9,2) = {'SCC Low Pass Filter Frequency'};
info_string(10,2) = {'Otolith Low Pass Filter Frequency'};
info_string(11,2) = {'Idiotropic Bias Value'};
info_string(14,2) = {'SCC Feedback Gain (kw)'};
info_string(15,2) = {'GIF Feedback Gain (kf)'};
info_string(16,2) = {'GIF Omega Feedback Gain (kfw)'};
info_string(17,2) = {'Acceleration Feedback Gain (ka)'};
info_string(18,2) = {'Omega GIF Feedback Gain (kwf)'};
info_string(21,2) = {'G Level'};
info_string(24,2) = {'Position LPF Frequency'};
info_string(25,2) = {'Velocity LPF Frequency'};
info_string(26,2) = {'Angular Velocity LPF Frequency'};
info_string(29,2) = {'Linear Position Gain (kxvv)'};
info_string(30,2) = {'Linear Velocity Gain (kxdotva)'};
info_string(31,2) = {'Angular Velocity Gain (kwvw)'};
info_string(32,2) = {'Tilt Angle Gain (kgvg)'};



%C Column
info_string(2,3) = {handles.file_to_open};
info_string(3,3) = {handles.duration};
info_string(4,3) = {handles.sample_rate};
info_string(7,3) = {handles.tau_scc_value};
info_string(8,3) = {handles.tau_a_value};
info_string(9,3) = {handles.f_scc};
info_string(10,3) = {handles.f_oto};
info_string(11,3) = {handles.w};
info_string(14,3) = {handles.kww(1)};
info_string(15,3) = {handles.kfg(1)};
```

```
info_string(16,3) = {handles.kfw(1)};
info_string(17,3) = {handles.kaa(1)};
info_string(18,3) = {handles.kwg(1)};
info_string(21,3) = {handles.g};
info_string(24,3) = {handles.f_visX};
info_string(25,3) = {handles.f_visV};
info_string(26,3) = {handles.f_visO};
info_string(29,3) = {handles.kxvv(1)};
info_string(30,3) = {handles.kxdotva(1)};
info_string(31,3) = {handles.kwvw(1)};
info_string(32,3) = {handles.kgvg(1)};

%D Column
info_string(3,4) = {'sec'};
info_string(4,4) = {'Hz'};
info_string(7,4) = {'sec'};
info_string(8,4) = {'sec'};
info_string(9,4) = {'Hz'};
info_string(10,4) = {'Hz'};
info_string(21,4) = {'G'};
info_string(24,4) = {'Hz'};
info_string(25,4) = {'Hz'};
info_string(26,4) = {'Hz'};

% Write Entire String
xlswrite(file_export,info_string,'File Data','A1');


%Angular Velocity
ang_info_string(1,1) = {'Angular Velocity Data'};
ang_info_string(2,1) = {'omega x'};
ang_info_string(2,2) = {'omega y'};
ang_info_string(2,3) = {'omega z'};
ang_info_string(1,5) = {'Estimated Angular Velocity Data'};
ang_info_string(2,5) = {'omega x est'};
ang_info_string(2,6) = {'omega y est'};
ang_info_string(2,7) = {'omega z est'};
xlswrite(file_export,handles.t_s,'Angular Velocity','A3');
xlswrite(file_export,ang_info_string,'Angular Velocity','B1');
xlswrite(file_export,handles.omega_head,'Angular Velocity','B3');
omega_est_plot = handles.omega_est*180/pi;
xlswrite(file_export,omega_est_plot,'Angular Velocity','F3');

%Angular Acceleration
ang_acc_info_string(1,1) = {'Angular Acceleration Data'};
ang_acc_info_string(2,1) = {'omega dot x'};
ang_acc_info_string(2,2) = {'omega dot y'};
ang_acc_info_string(2,3) = {'omega dot z'};
ang_acc_info_string(1,5) = {'Estimated Angular Acceleration Data'};
ang_acc_info_string(2,5) = {'omega dot x est'};
ang_acc_info_string(2,6) = {'omega dot y est'};
ang_acc_info_string(2,7) = {'omega dot z est'};
xlswrite(file_export,handles.t_s,'Angular Acceleration','A3');
xlswrite(file_export,ang_acc_info_string,'Angular Acceleration','B1');
xlswrite(file_export,handles.omega_dot_head,'Angular Acceleration','B3');
xlswrite(file_export,handles.omega_dot_est,'Angular Acceleration','F3');
```

```
%Linear Acceleration
lin_info_string(1,1) = {'Linear Acceleration Data'};
lin_info_string(2,1) = {'a x'};
lin_info_string(2,2) = {'a y'};
lin_info_string(2,3) = {'a z'};
lin_info_string(1,5) = {'Estimated Linear Acceleration Data'};
lin_info_string(2,5) = {'a x est'};
lin_info_string(2,6) = {'a y est'};
lin_info_string(2,7) = {'a z est'};
xlswrite(file_export,handles.t_s,'Linear Acceleration','A3');
xlswrite(file_export,lin_info_string,'Linear Acceleration','B1');
xlswrite(file_export,handles.a_head,'Linear Acceleration','B3');
xlswrite(file_export,handles.a_est,'Linear Acceleration','F3');

%Displacement
disp_info_string(1,1) = {'Displacement Data'};
disp_info_string(2,1) = {'d x'};
disp_info_string(2,2) = {'d y'};
disp_info_string(2,3) = {'d z'};
disp_info_string(1,5) = {'Estimated Displacement Data'};
disp_info_string(2,5) = {'d x est'};
disp_info_string(2,6) = {'d y est'};
disp_info_string(2,7) = {'d z est'};
xlswrite(file_export,handles.t_s,'Displacement','A3');
xlswrite(file_export,disp_info_string,'Displacement','B1');
xlswrite(file_export,handles.x_in,'Displacement','B3');
xlswrite(file_export,handles.x_est,'Displacement','F3');

%GIF
gif_info_string(1,1) = {'GIF Data'};
gif_info_string(2,1) = {'GIF x'};
gif_info_string(2,2) = {'GIF y'};
gif_info_string(2,3) = {'GIF z'};
gif_info_string(1,5) = {'Estimated GIF Data'};
gif_info_string(2,5) = {'GIF x est'};
gif_info_string(2,6) = {'GIF y est'};
gif_info_string(2,7) = {'GIF z est'};
xlswrite(file_export,handles.t_s,'GIF','A3');
xlswrite(file_export,gif_info_string,'GIF','B1');
xlswrite(file_export,handles.gif_head,'GIF','B3');
xlswrite(file_export,handles.gif_est,'GIF','F3');

%G
g_info_string(1,1) = {'Gravity Data'};
g_info_string(2,1) = {'g x'};
g_info_string(2,2) = {'g y'};
g_info_string(2,3) = {'g z'};
g_info_string(1,5) = {'Estimated Gravity Data'};
g_info_string(2,5) = {'g x est'};
g_info_string(2,6) = {'g y est'};
g_info_string(2,7) = {'g z est'};
xlswrite(file_export,handles.t_s,'Gravity','A3');
xlswrite(file_export,g_info_string,'Gravity','B1');
xlswrite(file_export,handles.g_head,'Gravity','B3');
xlswrite(file_export,handles.g_est,'Gravity','F3');
```

```
%SVV and  Tilt
tilt_info_string(1,1) = {'Subjective Visual Vertical Data'};
tilt_info_string(2,1) = {'Actual Vertical'};
tilt_info_string(2,2) = {'SVV'};
tilt_info_string(1,5) = {'Tilt (G) Angle Data'};
tilt_info_string(2,5) = {'Tilt Angle'};
tilt_info_string(2,6) = {'Estimated Tilt Angle'};
xlswrite(file_export,handles.t_s,'SVV and Tilt','A3');
xlswrite(file_export,tilt_info_string,'SVV and Tilt','B1');
xlswrite(file_export,handles.plot_SVV,'SVV and Tilt','B3');
xlswrite(file_export,handles.plot_tilt,'SVV and Tilt','F3');

%Linear Velocity
lin_vel_info_string(1,1) = {'Linear Velocity Data'};
lin_vel_info_string(2,1) = {'V x'};
lin_vel_info_string(2,2) = {'V y'};
lin_vel_info_string(2,3) = {'V z'};
lin_vel_info_string(1,5) = {'Estimated Linear Velocity Data'};
lin_vel_info_string(2,5) = {'V x est'};
lin_vel_info_string(2,6) = {'V y est'};
lin_vel_info_string(2,7) = {'V z est'};
xlswrite(file_export,handles.t_s,'Linear Velocity','A3');
xlswrite(file_export,lin_vel_info_string,'Linear Velocity','B1');
xlswrite(file_export,handles.lin_vel,'Linear Velocity','B3');
xlswrite(file_export,handles.lin_vel_est,'Linear Velocity','F3');

%Euler Angles
euler_info_string(1,1) = {'Euler Angle Data'};
euler_info_string(2,1) = {'Roll'};
euler_info_string(2,2) = {'Pitch'};
euler_info_string(2,3) = {'Yaw'};
euler_info_string(1,5) = {'Estimated Euler Angle Data'};
euler_info_string(2,5) = {'Roll est'};
euler_info_string(2,6) = {'Pitch est'};
euler_info_string(2,7) = {'Yaw est'};
xlswrite(file_export,handles.t_s,'Euler Angles','A3');
xlswrite(file_export,euler_info_string,'Euler Angles','B1');
xlswrite(file_export,handles.euler_head,'Euler Angles','B3');
xlswrite(file_export,handles.euler_est,'Euler Angles','F3');

set([handles.wait_text],'Visible','off');
set([handles.plot_pushbutton],'Enable','on');
set([handles.gif_visual_pushbutton],'Enable','on');
set([handles.start_simulation],'Enable','on');
set([handles.export_Data],'Enable','on');
set([handles.VR_pushbutton],'Enable','on');
set(handles.start_simulation,'CData',imread('startON.jpg'));
set(handles.export_Data,'CData',imread('exportON.jpg'));
set(handles.plot_pushbutton,'CData',imread('plotON.jpg'));
set(handles.gif_visual_pushbutton,'CData',imread('3dON.jpg'));
set(handles.VR_pushbutton,'CData',imread('vrON.jpg'));
%**************************************************************************
```

APPENDIX D. 3D VISUALIZATION (PLOTTOOLSAZI.M) MATLAB CODE

```
% ******** COPYRIGHT MASSACHUSETTES INSTITUTE OF TECHNOLOGY (c) 2008 ********
% Contact: Charles Oman (COMAN@MIT.EDU) or Michael Newman (M_NEWMAN@MIT.EDU)*
%*************************************************************************


% *************** Begin initialization code - DO NOT EDIT ******************
function varargout = plotToolsAzi(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @plotToolsAzi_OpeningFcn, ...
                   'gui_OutputFcn',  @plotToolsAzi_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
%*************************************************************************



%************ Executes just before plottools is made visible *****************
function plotToolsAzi_OpeningFcn(hObject, eventdata, handles, varargin)

movegui(gcf,'center');
% Set background image and properties
ha = axes('units','normalized','position',[0 0 1 1]);
uistack(ha,'bottom');
Image = imread('plotback.jpg');
plot_background = imagesc(Image);
set(ha,'handlevisibility','off','visible','off')


handles.output = hObject;
g_head = evalin('base', 'g_head');
g_est = evalin('base', 'g_est');
t_s = evalin('base', 'sim_time');
azimuth_head = evalin('base', 'azimuth_head');
azimuth_est = evalin('base', 'azimuth_est');

handles.g_head = g_head;
handles.g_est = g_est;
handles.t_s = t_s;
handles.azimuth_head = azimuth_head;
handles.azimuth_est = azimuth_est;
```

```
axes(handles.axes_3d_plot)
handles.line1 = line([0,g_head(1,1)],[0,g_head(1,2)],[0,g_head(1,3)]);
handles.line2 = line([0,g_est(1,1)],[0,g_est(1,2)],[0,g_est(1,3)]);

axis tight
view([30 15]);
grid('on');
title('Gravity','FontWeight','bold','FontSize',12)
set(gca,'nextplot','replacechildren','XLim',[-1 1],'YLim',[-1 1],'ZLim',[-1
1]);
set(handles.line1,'Color',[1 0 0],'Erase','xor','LineWidth',3);
set(handles.line2,'Color',[0 0 1],'Erase','xor','LineWidth',3);
line([-1 1],[0 0],[0 0],'LineWidth',3,'Color',[0,0,0]);
line([0 0],[-1 1],[0 0],'LineWidth',3,'Color',[0,0,0]);
line([0 0],[0 0],[-1 1],'LineWidth',3,'Color',[0,0,0]);

axes(handles.axes_viewer)
plot1 = plot(handles.t_s,handles.g_est);
xlim([0 max(handles.t_s)]);
ylim([-1 1]);
set(plot1(1),'Color',[1 0 0]);
set(plot1(2),'Color',[0 0 1]);
set(plot1(3),'Color',[0 0.498 0]);
handles.line_track = line([0 0],[-1 1],[0 0],'LineWidth',2,'Color',[0,0,0]);

axes(handles.axes_azimuth)

% Convert from polar to cartesian coordinates (because matlab polar
% plotting is pretty horrible

rho = ones(length(azimuth_head),1);
[X,Y] = pol2cart(azimuth_head,rho);
handles.azimuth_head_cart = [X,Y];

rho = ones(length(azimuth_est),1);
[X,Y] = pol2cart(azimuth_est,rho);
handles.azimuth_est_cart = [X,Y];


% Draw Unit Circle

x=-1:0.01:1;
lim = length(x);
for i = 1:lim;
    y(i,1) = sqrt(1-x(i)^2);
    y(i,2) = -sqrt(1-x(i)^2);
end
plot1 = plot(x,y);
set(plot1(1),'Color',[0 0 0],'LineWidth',1);
set(plot1(2),'Color',[0 0 0],'LineWidth',1);

% Draw Degree Mark lines every 45 degrees
```

```
deg_90 = line([0,0],[-1,0]);
set(deg_90, 'Color',[0,0,0],'LineStyle',':');
deg_45 = line([.7071,0],[-.7071,0]);
set(deg_45, 'Color',[0,0,0],'LineStyle',':');

deg_90 = line([0,-1],[0,0]);
set(deg_90, 'Color',[0,0,0],'LineStyle',':');
deg_45 = line([-.7071,0],[-.7071,0]);
set(deg_45, 'Color',[0,0,0],'LineStyle',':');

deg_90 = line([0,0],[0,1]);
set(deg_90, 'Color',[0,0,0],'LineStyle',':');
deg_45 = line([-.7071,0],[.7071,0]);
set(deg_45, 'Color',[0,0,0],'LineStyle',':');

deg_90 = line([0,1],[0,0]);
set(deg_90, 'Color',[0,0,0],'LineStyle',':');
deg_45 = line([.7071,0],[.7071,0]);
set(deg_45, 'Color',[0,0,0],'LineStyle',':');

grid('off');
box('on');
set(handles.axes_azimuth,'YTick',zeros(1,0),'XTick',zeros(1,0),'YColor',[1 1
1],'XColor',[1 1 1],'Color',[1 1 1]);

handles.azimuth_line =
line([0,handles.azimuth_head_cart(1,1)],[0,handles.azimuth_head_cart(1,2)]);
handles.azimuth_line_est =
line([0,handles.azimuth_est_cart(1,1)],[0,handles.azimuth_est_cart(1,2)]);

set(handles.azimuth_line,'Color',[1 0 0],'LineWidth',3);
set(handles.azimuth_line_est,'Color',[0 0 1],'LineWidth',3);

xlim([-1 1]);
ylim([-1 1]);

guidata(hObject, handles);
%*********************************************************************



% ************* Create the Output VARARGOUT Function DO NOT EDIT ************
function varargout = plotToolsAzi_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%*********************************************************************



% ***** Play, Pause and Close Pushbuttons. Create Plots. Plot Data **********
function visual_pushbutton_Callback(hObject, eventdata, handles)
set([handles.visual_pushbutton], 'Enable','off');
set([handles.pause_toggle], 'Enable','on');

i = 1;
while i <= length(handles.g_head)
```

```
    if get(handles.close_pushbutton,'Value') ==
get(handles.close_pushbutton,'Max')
        close Visualization;
        break;
    end

    axes(handles.axes_3d_plot)

set(handles.line1,'XData',[0,handles.g_head(i,1)],'YData',[0,handles.g_head(i
,2)],'ZData',[0,handles.g_head(i,3)]);

set(handles.line2,'XData',[0,handles.g_est(i,1)],'YData',[0,handles.g_est(i,2
)],'ZData',[0,handles.g_est(i,3)]);

    axes(handles.axes_viewer)

set(handles.line_track,'XData',[(max(handles.t_s)*i)/length(handles.t_s),(max
(handles.t_s)*i)/length(handles.t_s)],'YData',[-1,1],'ZData',[0,0]);

    axes(handles.axes_azimuth)

set(handles.azimuth_line,'XData',[0,handles.azimuth_head_cart(i,1)],'YData',[
0,handles.azimuth_head_cart(i,2)]);

set(handles.azimuth_line_est,'XData',[0,handles.azimuth_est_cart(i,1)],'YData
',[0,handles.azimuth_est_cart(i,2)]);

    drawnow

    pause(0.01)

    if i >= (length(handles.g_head) -
floor(length(handles.g_est)/(10*max(handles.t_s))))
        set([handles.visual_pushbutton], 'Enable','on');
    end

    i = i + floor(length(handles.g_est)/(5*max(handles.t_s)));
end

function pause_toggle_Callback(hObject, eventdata, handles)
button_state = get(handles.pause_toggle,'Value');
    if button_state == get(handles.pause_toggle,'Max')
        uiwait;
    elseif button_state == get(handles.pause_toggle,'Min')
        uiresume;
    end

function close_pushbutton_Callback(hObject, eventdata, handles)

 if get(handles.pause_toggle,'Value') == get(handles.pause_toggle,'Max')
        uiresume;
 end
 if get(handles.visual_pushbutton,'Value') ==
get(handles.visual_pushbutton,'Min')
```

```
        close Visualization;
 end
%*************************************************************************

% ******** Change the View of the 3D plot, between Planes and 3D ************
function YZ_pushbutton_Callback(hObject, eventdata, handles)
axes(handles.axes_3d_plot)
view([90 0]);

set([handles.X],'Visible','off');
set(handles.X,'String','X');
set([handles.Y],'Visible','off');
set(handles.Y,'String','Y');
set([handles.Z],'Visible','on');
set(handles.Z,'String','Z');
set([handles.Bot_Axis],'Visible','on');
set(handles.Bot_Axis,'String','Y');

function XY_pushbutton_Callback(hObject, eventdata, handles)
axes(handles.axes_3d_plot)
view([0 90]);
set([handles.X],'Visible','off');
set(handles.X,'String','X');
set([handles.Y],'Visible','off');
set(handles.Y,'String','Y');
set([handles.Z],'Visible','on');
set(handles.Z,'String','Y');
set([handles.Bot_Axis],'Visible','on');
set(handles.Bot_Axis,'String','X');

function XZ_pushbutton_Callback(hObject, eventdata, handles)
axes(handles.axes_3d_plot)
view([0 0]);

set([handles.X],'Visible','off');
set(handles.X,'String','X');
set([handles.Y],'Visible','off');
set(handles.Y,'String','Y');
set([handles.Z],'Visible','on');
set(handles.Z,'String','Z');
set([handles.Bot_Axis],'Visible','on');
set(handles.Bot_Axis,'String','X');

function normal_pushbutton_Callback(hObject, eventdata, handles)
axes(handles.axes_3d_plot)
view([30 15]);
set([handles.X],'Visible','on');
set(handles.X,'String','X');
set([handles.Y],'Visible','on');
set(handles.Y,'String','Y');
set([handles.Z],'Visible','on');
set(handles.Z,'String','Z');
set([handles.Bot_Axis],'Visible','off');
set(handles.Bot_Axis,'String',' ');
%*************************************************************************
```

APPENDIX E. BLOCK DIAGRAM (OBSERVERMODEL.MDL) SIMULINK MODEL

For details see Figure 4.